XC: Exploring Quantitative Use Cases for Explanations in 3D Object Detection

by

Sunsheng Gu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Explainable AI (XAI) methods are frequently applied to obtain qualitative insights about deep models' predictions. However, such insights need to be interpreted by a human observer to be useful. In this thesis, we aim to use explanations directly to make decisions without human observers. We adopt two gradient-based explanation methods, Integrated Gradients (IG) and backprop, for the task of 3D object detection. Then, we propose a set of quantitative measures, named Explanation Concentration (XC) scores, that can be used for downstream tasks. These scores quantify the concentration of attributions within the boundaries of detected objects. We evaluate the effectiveness of XC scores via the task of distinguishing true positive (TP) and false positive (FP) detected objects in the KITTI and Waymo datasets. The results demonstrate improvement of more than 100% on both datasets compared to other heuristics such as random guesses and number of LiDAR points in bounding box, raising confidence in XC's potential for application in more use cases. Our results also indicate that computationally expensive XAI methods like IG may not be more valuable when used quantitatively compared to simpler methods. Moreover, we apply loss terms based on XC and pixel attribution prior (PAP), which is another qualitative measure for attributions, to the task of training a 3D object detection model. We show that performance boost is possible as long as we select the right subset of predictions for which the attribution-based losses are applied.

Acknowledgements

I would like to thank Dr. Krzysztof Czarnecki for granting me the opportunity to work in the WISE lab as a masters student and guiding me through my two-year journey of pursuing this degree. Special thanks goes to Dr. Vahdat Abdelzad who had helped me generate new ideas on numerous occasions and helped me see flaws and gaps in my experiments, thus motivating me to dig further and to produce better results. I would also like to express my gratitude to Chengjie Huang, who had frequently helped me with debugging and technical difficulties.

I would like to thank Huawei Noah's Ark Lab, Canada, for sponsoring my work.

I also thank Dr. Sean Sedwards, Dr. Rick Salay, Van Duong Nguyen, and Matthew Pitropov for their support and feedback.

Dedication

This is dedicated to my family and friends who have supported and encouraged me in difficult times.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Recent development in deep neural networks (DNNs) has led to the state of the art performance on 2D [16, 18, 32, 39, 40] and 3D [25, 47, 62] object detection tasks. However, despite of the improvement in model performance, the lack of model interpretability remains a significant drawback. This issue has sparked interest in *explainable artificial intelligence* (XAI). The primary goal of these XAI methods is to uncover the logic behind the models' decisions [29].

Several recent methods have been proposed to generate interpretable visual explanations [44, 48, 51, 53, 61]. They are usually analyzed by a human observer to understand the model's reasoning for some particular decisions. Hence, explanations are very useful for debugging and diagnosis. However, it would be impossible for a human to analyze each instance in a large dataset. To analyze explanations on a large scale, it is necessary to derive quantitative measures from the explanations. There have been a few works in this direction [11, 42, 43], one notable measure is pixel attribution prior [11], which measures the smoothness of the attribution map. But the input data experimented with in these works are 2D image, text, or tabular data. We have not found any experiments on explanations-related quantitative measures applied to LiDAR point cloud input or to the task of object detection. Our motivation is to fill in the gap by exploring quantitative usage of explanations for LiDAR-based object detection. We choose to focus on 3D LiDAR data mainly due to its relevance to robot perception, specifically for the task of automated driving.

1.2 Overview

We study explanations in the form of attribution maps for a 3D object detector named PointPillars [25] and use them for the problem of distinguishing true positive (TP) vs. false positive (FP) predictions. Attributions, in the context of XAI, denote the influence of input features on model output. We generate attribution maps using two XAI methods: 1) Integrated Gradients (IG) [53], selected for its axiomatic properties; 2) backpropagation [48], selected for its low computational cost. Our main contributions are [1]:

- We demonstrate that quantitative usage of explanation for 3D object detection is a promising direction for future research: We propose a set of quantitative metrics called *Explanation Concentration* (XC), which measures the concentration of attributions within each predicted object. XC can be used to classify the TP vs. FP predictions effectively, often achieving more than 100% improvement compared to simple heuristics such as number of LiDAR points within a predicted box.
- We discover that XC scores derived from backpropagation can perform better than those derived from IG.
- We propose a new score that can identify TP vs. FP predictions better than the individual XC scores and object class score. This score is generated by combining the XC scores with object class score using a MLP.
- We design loss functions based on XC and pixel attribution prior and apply them when training PointPillars. We discover that in some cases, the model trained with attribution related losses can notably outperform the model trained without such losses.

Chapter 2 briefly introduces related work on XAI methods, quantitative usages of XAI, LiDAR-based object detection, and false positive detection. Chapter 3 then describes how we obtain feature attributions for PointPillars and how the XC scores are computed from the attributions. Chapter 4 describes the experiments on using XC derived from IG and backpropagation attributions to help identify TP vs. FP predictions, as well as combining the XC scores with class score to further improve performance. And lastly in Chapter 5 we present results on using attribution-related loss functions to train PointPillars.

[1] Our code is available at https://github.com/SunshengGu/XC_eval_pcdet.

Chapter 2

Related work

2.1 Explanation methods

As listed in the Cambridge Dictionary of English Language, the term "explanation" is defined as "*the details or reasons that someone gives to make something clear or easy to understand*"[55]. Arrieta *et al.* [1] rephrased this definition in the context of machine learning as "*the details or reasons a model gives to make its functioning clear or easy to understand*". Interpretability, on the other hand, can be defined as "*the ability to explain or to provide the meaning in understandable terms to a human*" [1]. Some simple models, such as logistic regression or decision trees, are highly interpretable. It can be said that these models explain themselves. A human observer would not need any extra algorithms/tools to understand the model's reasoning. However, more complex models with millions of parameters, such as deep neural networks, can not be interpreted by a human being. The model's parameters cannot offer an explanation for its reasoning that can be understand by humans. This is where Explainable AI (XAI) methods can be helpful. There are many methods are designed to reveal complex models' reasoning easily understandable by human observers. Details will be provided in the following sections.

2.1.1 Categorization of explanation methods

This section discusses the categorization of XAI methods. XAI approaches can be categorized along four dimensions [1]:

1. Model dependency

2. Level of explanation
3. Time of explanation
4. Methodology

Explanation approaches can be categorized by dependency on the model being explained. When an explanation approach does not require any knowledge of the model besides model inputs and outputs, then the approach is said to be "model-agnostic". When an explanation approach relies on access to model parameters to generate explanations, such an approach is called "model-dependent".

The level of explanation can also be used to categorize different approaches. Explanations can be at the model or class level or at the instance level. At the model level, the reasoning of the model towards a specific class or classes is explained. This requires global understanding of the model and is very difficult to achieve. At the instance level, the reasoning of the model towards a specific input and output pair is explained. Note that all the explanation methods presented in Section 2.1.2 to Section 2.1.4 are generating instance level explanation, which is reflective of the current state of the field.

Explanations can also be generated at different points in time. Very frequently, explanations are generated in a post-hoc fashion, i.e., explaining the model's reasoning after the model has completed its task. In this case, the explanation process is not part of the model. In some other cases, explanations are generated in an intrinsic fashion, i.e., explanations are generated by the model itself as it is performing its primary task (e.g., classification or regression). Such approaches are useful when explanations can help the model perform better on its primary task.

We can also categorize the explanation methods based on the way how the explanations are generated. One could use a separate simpler model (i.e., the surrogate model) to generate explanations for a more complex model, or apply perturbation to model inputs and analyze changes in model outputs (perturbation analysis), or generate explanations by computing the gradient of model output with respect to model input (gradient-based explanations). The following sections discuss each of these mythologies in details. The explanation methods experimented in this thesis, backprop and Integrated Gradients, are all gradient-based methods.

2.1.2 Surrogate model

One method for creating explanations is to learn a surrogate model. The surrogate model is more interpretable than the original model, but generates similar output given the same

input. The original model could be a convolutional neural network (CNN), whereas the surrogate model would be a linear model or decision tree. One famous work applying such technique is Local Interpretable Model-agnostic Explanations (LIME) [41]. The intuition behind the LIME technique is demonstrated in Figure 2.1. The key idea is to learn a locally faithful linear classifier as a surrogate model for the original model. In Figure 2.1, the boundary between the blue and pink regions represents the decision boundary learned by the original model. The bold red cross represents the instance to be explained, the smaller red crosses represent samples from the same class, and the dark blue dots are samples from another class. The size of a cross/dot represents its weight, the closer it is to the instance of interest (bold red cross), the more important that sample is. Then from this weighted set of samples, a linear classifier is learned, signified by the dashed line. This linear classifier roughly coincides with the original model's decision boundary in the instance's vicinity, and the goal of learning a locally faithful surrogate model has been achieved. To obtain the instances in the neighborhood of the input image to learn the linear surrogate model and generate interpretable explanations, LIME converts the input image into a vector of superpixels. Then many instances close to the input image can be generated by simply graying out or retaining these superpixels. Explanations are obtained by producing instances that corresponds to different predictions, as shown in Figure 2.2.

LIME provides instance level explanation by design, although it could be used to generate model level explanation by combining surrogate models learned from many different instances across the input distribution. LIME is model-agnostic and generates post-hoc explanations.

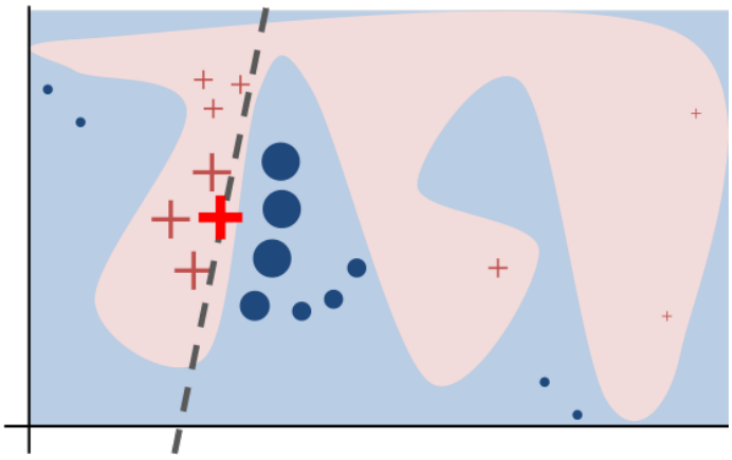
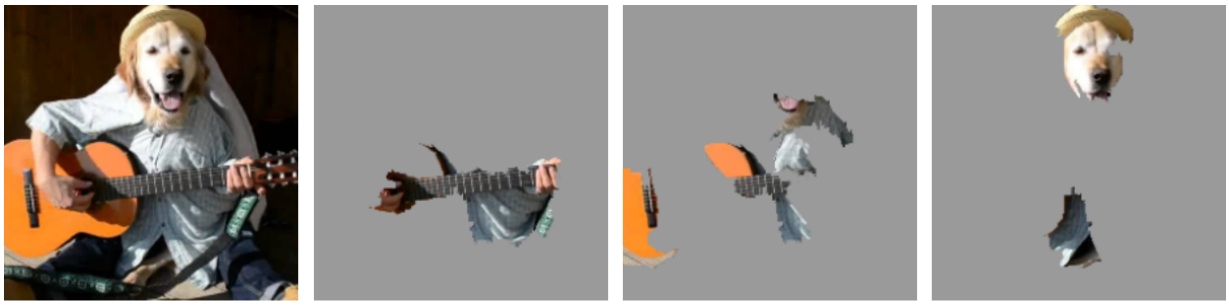


Figure 2.1: Illustration of LIME's surrogate model generation process [41].



(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figure 2.2: LIME's explanation for different classes [41].

2.1.3 Perturbation analysis

Explanations may also be generated by introducing perturbations in the input. One common approach is to partially occlude an input to learn regional importance. Zeiler and Fergus [61] presented a partial occlusion analysis in the form of heatmap. Their approach is illustrated in Figure 2.3. The input image contains a Pomeranian dog. A smaller square grey box is overlaid onto the input image to partially occlude it. The partially occluded image is fed through the classification model and the class probability for Pomeranian is obtained. The grey box is moved across the input image according to a certain step size, the model performs classification at each of these steps. The class probability for the correct class is plotted as the function of the grey box center location. The end result is a heatmap. Regions with low scores are recognized as more important for the class of interest. From the given example, one can observe that when the face of the dog is occluded, there is a significant drop in class probability for "Pomeranian", indicating that the dog's face is important for correctly classifying it, which is a quite reasonable explanation. This partial occlusion scheme generates explanation in a post-hoc fashion, and is model-agnostic.

2.1.4 Gradient-based methods

There are an abundance of gradient-based approaches for assigning importance to input features. The values indicating feature importance are usually referred to as *attributions*, and the attributions form an *attribution map* that is usually the same size as the input (see Figure 2.4 for examples). The intuition behind these approaches are similar to that of backpropagation. In the context of deep learning, backpropagation usually refers to the process of computing the gradient of the loss function with respect to model parameters, so that the parameters can be adjusted to improve model performance. In the context

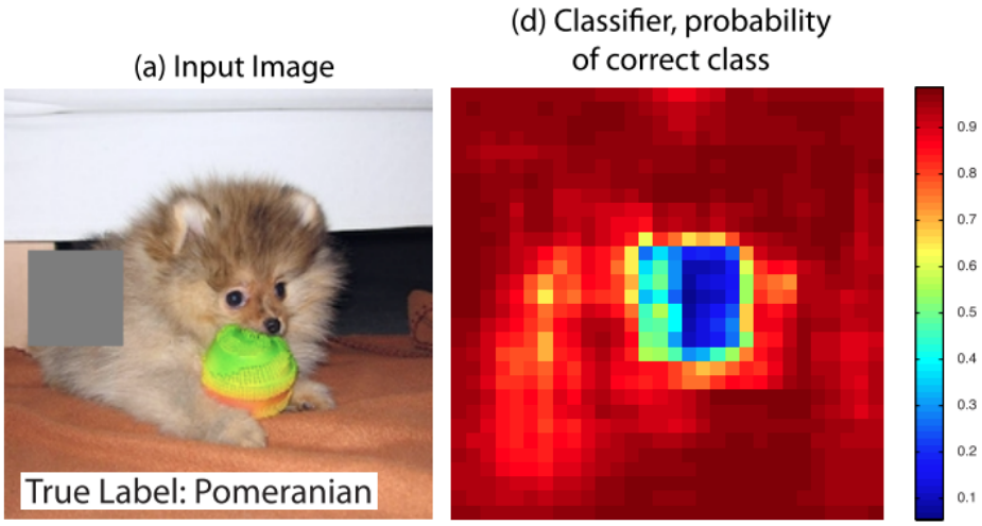


Figure 2.3: Partial occlusion analysis for a dog image [61].

of XAI, backpropagation can be modified to compute the gradient of an output neuron with respect to input features [61]. In classification, an output neuron represents a specific class. Computing the gradient of this neuron with respect to an input pixel would reveal the importance of this specific pixel to this specific class.

Besides this modified version of backpropagation, there are two other similar approaches: deconvolutional neural networks (DeconvNet, proposed by Zeiler and Fergus [61] as a visualization technique and refined by Springenberg *et al.* [51]) and Guided-backpropagation [51]. Assuming ReLU activation function is applied, backprop would zero out gradients for neurons with negative values in the forward pass, and DeconvNet zero out negative gradients in the backward pass. Guided-backprop combines the two methods and filters out negative values in both forward and backward passes. The resulting attribution map from Guided-backprop would highlight features that positively influence the model output.

Integrated gradients (IG) [53] is a more sophisticated gradient-based approach for assigning attributions to inputs. It is designed based on two axioms which the authors claim are crucial for generating valid explanations: *sensitivity* and *implementation invariance*. Sensitivity has two components: 1) input features that influence the model output are guaranteed to have nonzero attributions, and 2) input features that do not influence the model output are guaranteed to have zero attributions. Implementation invariance, on the other hand, ensures that the explanation method generates the same explanations for two

functionally equivalent models. Two models are said to be functionally equivalent if, given the same input, they always generate identical output. Most gradient-based explanation approaches satisfy implementation invariance due to chain rule: gradient of output with respect to input equals to the product of the intermediate gradients. But these gradient-based approaches fail to satisfy sensitivity, because activation functions such as ReLU will filter out negative values in the forward pass. As a result, the explanation will not capture all influences in the input image.

IG ensures sensitivity by computing the path gradient along a straight line from the baseline x' to the input x. The baseline x' is an input which produces 0 output when fed through the model. For perception, the baseline is a black image with all feature values equals to zero. Integrated gradient is defined below:

$$\text{IntegratedGrads}_{\,i}(x) ::= (x_i - x'_i) \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha \tag{2.1}$$

Where $F(input)$ represents model output for a certain input. In this way, the attribution values are scaled by the difference between the input feature value and the baseline value (0 in this case), and features identical to the baseline are guaranteed to have zero attributions. Note that the subscript i signifies a specific feature in the input, and integrated gradient is computed in an element-wise fashion. The authors pointed out that the sum of all the individual i^{th} integrated gradients is the same as the difference in model outputs between the input and the baseline. Hence, all differences in outputs are attributed to differences in input features (i.e., pixels for images) and sensitivity is satisfied. In addition, since this approach is gradient based, implementation invariance is satisfied as well. The integral part in Equation (2.1) is approximated using Riemann sum by taking a certain number of steps from x' to x. The authors suggest taking 20 to 300 steps to achieve less than 5% error. The attribution maps generated from integrated gradients are compared with those generated from backpropagation in Figure 2.4. It is clear that integrated gradients precisely identifies pixels which are important for the class of interest, whereas backpropagation sometimes highlights unimportant pixels and ignores important ones due the fact that sensitivity is not satisfied.

Despite of IG's recent popularity, Erions *et al.* [11] challenge it by pointing out that IG's choice of a black image as baseline is not always valid. Take MNIST for example, if one flips the pixel values, then the digits patterns are represented by black pixels with zero values, and the background pixels would be white. In this case, the zero-valued black pixels should get some attribution values. Instead of picking a fixed baseline, the authors propose a new method named *Expected Gradients* (EG), where a certain number of baseline images

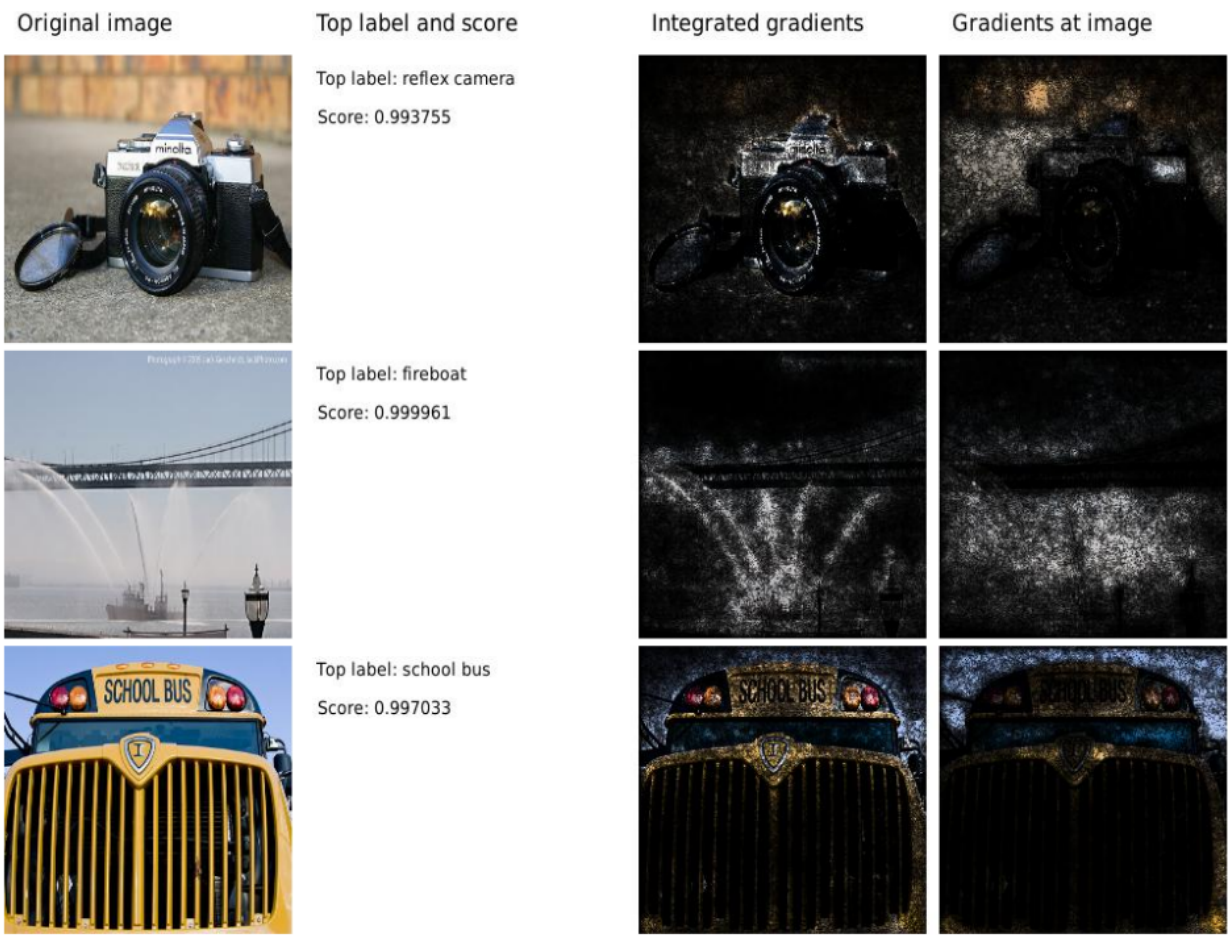


Figure 2.4: Pixel attribution for various classes generated by integrated gradients and backpropagation (i.e., gradients) [53].

are randomly sampled from the training set D and α is sampled from $U(0,1)$:

$$\text{ExpectedGradients}_{\ i}(x) := \mathop{\mathbb{E}}_{x' \sim D, \alpha \sim U(0,1)} \left[(x_i - x'_i) \times \frac{\delta f(x' + \alpha \times (x - x'))}{\delta x_i} \right] \tag{2.2}$$

The authors proposed eighteen evaluation metrics for attributions, and evaluated IG, EG, and backprop attributions generated from a toy dataset. EG slightly outperforms IG and greatly outperforms backprop in most of the metrics. They also trained models with a loss function based on EG attributions and improved performance on noise datasets, more details are provided in Section 2.2.2.

It is important to note that all gradient-based explanation approaches are post-hoc and model-dependent. Because backpropagation of gradients can only be performed after the model has processed the input and gradient values are dependent on the model parameters and architecture.

2.2 Quantitative Usage of Explanations

Explanations are usually analyzed qualitatively by human observers to help understand model reasoning and debug models. For example, overlaying attribution map on top of images can reveal which part of the input image influences model output. If an image classifier labels an image as "computer monitor" but the attribution map highlights the keyboard besides the monitor, then we know the model is focusing on the wrong thing. But explanations can also be used quantitatively. Several previous works [11, 42, 43] have explored the usage of explanations in training neural networks, the follow paragraphs discuss the key findings from these works.

2.2.1 Being right for the right reasons

Ross et al. [43] use explanations to 1) make the model ignore irrelevant features when prior knowledge about feature relevance is available and 2) make the model discover new rules for generating predictions when such prior is not available. They use gradient of the log of model output with respect to input features as attributions for these features. The prior knowledge about feature relevance is encoded as such:

$$A \in \{0,1\}^{N \times D} \tag{2.3}$$

where N is the number of training samples and D is the number of features per sample. Each entry in the matrix encode relevance of the corresponding feature, 1 means irrelevant,

and 0 means relevant. For example, if $A_{n,d} = 1$ then the d^{th} feature of the n^{th} training instance should have no influence on the model output.

Then the stage is set for a new loss function that can force the model to make decisions solely based on the relevant features:

$$L(\theta, X, y, A) = \underbrace{\sum_{n=1}^{N}\sum_{k=1}^{K} -y_{nk}\log\left(\hat{y}_{nk}\right)}_{\text{Right answers}} + \underbrace{\lambda_1 \sum_{n=1}^{N}\sum_{d=1}^{D}\left(A_{nd}\frac{\partial}{\partial x_{nd}}\sum_{k=1}^{K}\log\left(\hat{y}_{nk}\right)\right)^2}_{\text{Right reasons}} + \underbrace{\lambda_2 \sum_{i}\theta_i^2}_{\text{Regular}} \quad (2.4)$$

where K represents the number of possible classes, Θ represents model parameters, X and y represent training inputs and labels. In this way, nonzero attributions for irrelevant features are punished, and the model would learn to ignore such features in the input. The authors argue that this loss function can make the model make right decisions (achieved by the classification loss) for the right reasons (achieved by the attribution loss). They experimented this strategy on the Decoy MNIST dataset, a dataset similar to MNIST [26], but with gray patches of size 4×4 pixels randomly placed on one of the 4 corners of the input image. Without the attribution loss, the test accuracy is around 50%, but with the attribution loss penalizing attributions assigned to the gray patch pixels, the test accuracy reaches close to 100%.

Even when no prior knowledge about feature relevance is available, the attribution loss can still be used to help the model discover new rules iteratively. Initially, the matrix A is all zero. After the first training cycle, the features with high attribution values are marked as 1 in A, forcing the model to not rely on these features in the next training cycle. As a result, the new model would learn to generate predictions based on other features, hence learning new rules for making decisions. This process can be repeated multiple times to generate multiple models, each focusing on a different set of features. A domain expert can review these models and pick the best one.

2.2.2 Pixel attribution prior

Erion *et al.* [11] proposed a loss named "pixel attribution prior" (PAP) for image classification to make the model assign similar attributions to neighbouring pixels. The formulation is as follows:

$$\Omega_{\text{pixel}}\left(\Phi(\theta, X)\right) = \lambda \sum_{\ell}\sum_{i,j}\left|\phi_{i+1,j}^{\ell} - \phi_{i,j}^{\ell}\right| + \left|\phi_{i,j+1}^{\ell} - \phi_{i,j}^{\ell}\right| \quad (2.5)$$

where ℓ is the sample id, i and j represent pixel location, and ϕ represents attribution value. To train a model with this loss, one could replace the attribution loss term in Equation (2.4) with Equation (2.5). The intuition behind PAP is that neighbouring pixels in an image should have similar attributions. To a human observer, small variations in a few pixels should barely make any differences, but such variations can sometimes change model output drastically. The goal of PAP is to make the model more robust by training it to focus on larger patterns in the input image, rather than allowing a few pixels sway model decisions. Figure 2.5 shows how PAP changes the attribution map. One advantage of PAP compared to the method proposed by Ross *et al.* [43] is that it does not require any prior knowledge about feature relevance.

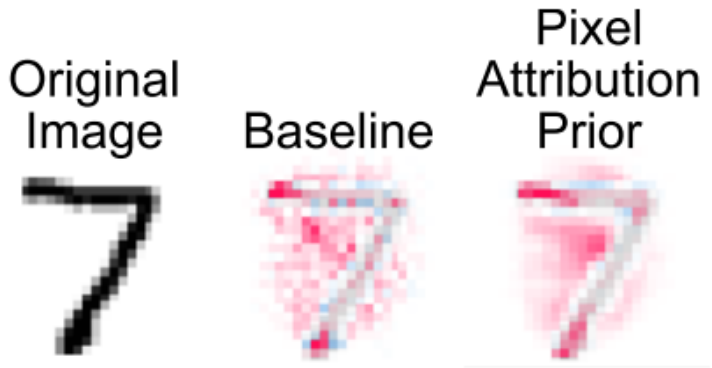


Figure 2.5: Attribution maps for digit 7 generated from a model trained without PAP and a model trained with PAP [11].

The authors injected Gaussian noise into MNIST [26] and CIFAR10 [23] to analyze the benefit offered by PAP. They compare performance of models trained with vs. without PAP. When training on the original datasets, the model trained with PAP is performing slightly worse than the other model. But when training on noisy datasets, the model trained with PAP outperforms the other model by large margins, sometimes leading to more than 50% improvement on test accuracy.

2.2.3 Other ways to use explanations quantitatively

Rieger *et al.* [42] also propose a method called CDEP to constrain attributions by defining a loss which is proportional to the difference between the attribution values and prior target attribution values. They are able to improve model performance on skin cancer detection,

as well as on variations of the MNIST dataset. Du *et al.* [10] propose a framework called CREX that can improve DNN performance on text data by forcing the model to generate explanations that conform to expert rationales (a subset of features recognized by domain experts as useful for predictions). Even when no such rationales are available, CREX can still generalize the model by ensuring sparse attributions.

In previous works on quantitative usages of explanations, the focus has always been to improve model performance. Similarly, in this thesis, we also attempt to improve object detector performance using quantitative measures of explanations (see Chapter 5). But we also explore a new use case: distinguishing true positive vs. false positive predictions using explanations (see Chapter 4).

2.3 LiDAR-based object detection

2.3.1 3D object detection models

Object detection is a more challenging task than image classification, as there can be multiple detected objects in an image or a LiDAR point cloud, and for each detection, both classification (for labelling the detection) and regression (for determining the detection box parameters) need to be performed. In 2D object detection, recent works mainly adopted two strategies: 1) A two-stage network where region proposals are generated in the first stage, then refined and filtered into box predictions in the second stage; R-CNN [16] and its derivatives [15, 17, 40] fall into this category. 2) A single-stage network where a fixed set of anchor boxes are filtered and refined to generate box predictions; YOLO [39] and SDD [32] fall into this category.

Since LiDAR point cloud is scattered in a 3D space, object detection using LiDAR data is a 3D perception problem. One popular technique for 3D object detection is to first voxelize the 3D point cloud into either 3D grids or 2D bird's eye view grids, then apply 3D or 2D convolutional layers to the voxel-wise features to learn higher level features. Techniques used for 2D object detection, such as region proposal and anchor boxes, can then be applied to the extracted feature maps. MV3D [5], Pixor [60], VoxelNet [62], SeCOND [59], and PointPillars [25] all partly adopted this strategy.

On the other hand, there are ways to extract features from point cloud without voxel feature extraction or any convolutional networks. PointNet [37] learns 3D features directly from point cloud and is invariant to point ordering, translation, and rotation. PointNet++ [38] further improves PointNet by capturing local structures. Recent 3D object detectors

have been using PointNet or PointNet++ as part of the network. For example, PointRCNN [47] uses PointNet++ as point feature extractor, PV-RCNN [46] uses PointNet++ to extract 3D convolutional features at different scales, and PointPillars [25] also uses PointNet as part of its voxel feature encoding process.

PointPillars' [25] superior performance on KITTI [14] LiDAR dataset and its fast inference speed (62Hz) make it a desirable choice for deployment on embedded hardware, such as in autonomous vehicles. It is a 3-stage network consists of a pillar feature net, a 2D CNN backbone, and an SSD [32] as the detection head. Hence, we choose PointPillars for our experiments.

2.3.2 Evaluation metrics for object detection

The performance of an object detector is most often evaluated in terms of *Average Precision* (AP). It evaluates a model's ability to correctly identify the cared objects without generating too many false predictions. To understand the AP metric, it is important to introduce the concept of precision and recall. In the context of object detection, the objects that are intended to be detected (also called "ground truth" objects) can be considered as the positive instances, and everything else (including background stuff like sky, road, and lawn) can be considered as the negative instances. Referring to Table 2.1, if a positive instance is correctly detected, then we have a true positive (TP), otherwise this object is a false negative (FN). If a detected object does not match any positive instances, then we have a false positive (FP). True negative (TN) is usually not evaluated for object detection, because there is an infinite number of negative instances (everything other than the cared objects). Note that one may also use the same confusion matrix (Table 2.1) to count the number of TP, FP, TN, FN in the context of binary classification, more details on that is provided in Section 4.1.

Intersection over union (IoU) is often used to match predicted objects with ground truth objects. IoU is the area (for 2D objects) or volume (for 3D objects) of the intersection of the predicted object and the ground truth object divided by the area or volume of their union. One can then set a threshold for IoU to decide if a prediction has a matching object. Precision (TP/(TP+FP)) reveals how much of the detected objects actually correspond to the ground truth objects, and recall (TP/(TP+FN)) reveals how much of the ground truth objects are correctly identified by the detector. A plot with precision as the y-axis and recall as the x-axis is called a precision-recall (PR) curve [8].

To plot the PR curve for an object detector, the final predictions are ranked based on object class score, the precision and recall values are then computed at different score

Table 2.1: Confusion matrix for two classes

	Positive instances	Negative instances
Predicted to be positive	TP	FP
Predicted to be negative	FN	TN

values. Say we have n predictions from a model, with scores $[S_0, S_1, ..., S_{n-1}]$, then we can step through each S_i as a threshold, and using only the predictions with score $>= S_i$ to compute and plot precision and recall at each step. Now the stage is set for computing average precision (AP).

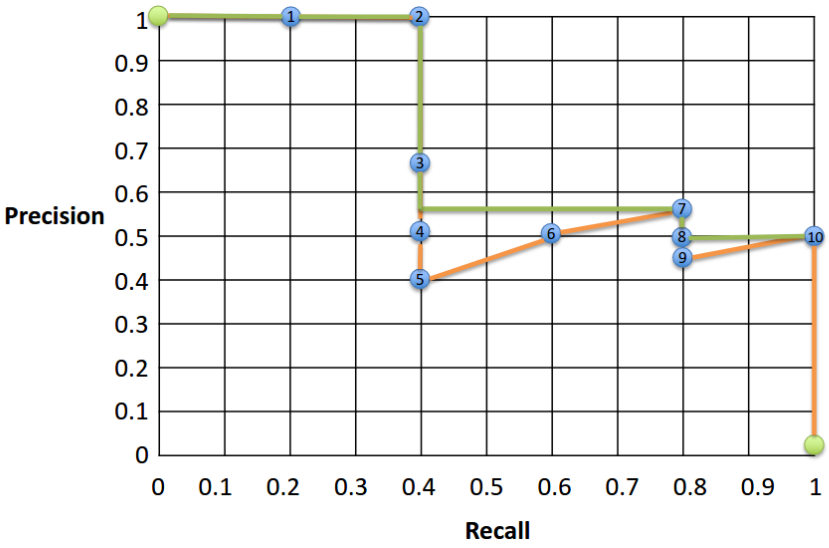


Figure 2.6: Smoothing a PR curve by monotone interpolation. The orange curve is the original PR curve, the green curve is the PR curve after interpolation [7].

AP is often obtained by first smoothing the PR curve and then averaging the precision values across a number of different recall values. By interpolating all precision values to match the greatest precision value to the right, small undulations in the PR curve caused by the ranking of predictions are filtered out (see Figure 2.6). The specific number of data points used varies for different datasets/models. For example, in Pascal VOC2008 [12], the average of precision values at 11 recall values (0, 0.1, ..., 0.9, 1) is reported. Whereas for the COCO dataset [31], up to 101 data points are used to obtain AP. Another way to

compute AP is to just compute the area under the smoothed PR curve, which is a strategy adopted by Pascal VOC2010 [13]. If a model achieves high AP, it is means this model is able to correctly identify most of the ground truth objects without generating too many FP. If AP is low, then that means the model misses too many ground truth objects and/or produces too many FP.

In object detection, the term mean AP (mAP) is often used interchangeably with AP. mAP originally means the mean of several different AP values. For example, one can evaluate AP using different IoU thresholds, and obtain the mean of these AP values. Or one can compute the mean of AP values obtained from different object classes. But later the meaning of mAP gets confused with AP and the true meaning of mAP or AP is often deduced from context [31].

2.4 False positive detection

Several attempts have been made to effectively identify false positive (FP) predictions. Hendrycks and Gimpel [19] demonstrated that the softmax class probability could effectively distinguish true positive (TP) and false positive (FP) predictions in various image classification, sentiment classification, and text categorization datasets, with performance far exceeding random guess. The intuition is that the model should have higher prediction confidence for the TP samples than for the FP ones.

On the other hand, Chen *et al.* [4] used a similar but more sophisticated approach to suppress false positives and uncover false negatives for pedestrian detection. They first run a pretrained model M_s and group the predictions into two sets, P containing the boxes with softmax score exceeding a high threshold h (0.95), and N containing the remaining boxes. Then they 1) retrain M_s to obtain a new model M_i and feed inputs from N through M_i, and 2) add a cluster of new predictions generated by M_i with high softmax score into P. In essence, the boxes are re-ranked based on softmax score in each iteration. The process is repeated until $|P|$ meets a preset threshold. Their method was proven to be effective in improving model performance on KITTI and Cityscapes [6].

Besides the two above mentioned works which applied techniques specifically for detecting/suppressing FPs, there are some other works focusing on handling adversarial attacks or detecting out-of-distribution (OOD) samples. The techniques proposed in these works might also be applied to detect FPs. Some notable works are feature squeezing [57], LID [34], and GraN [33] for detecting adversarial samples, ODIN [30] for OOD detection, and [28] who used Mahalanobis distance to detect both adversarial and OOD detection.

Chapter 3

Explanation Concentration (XC)

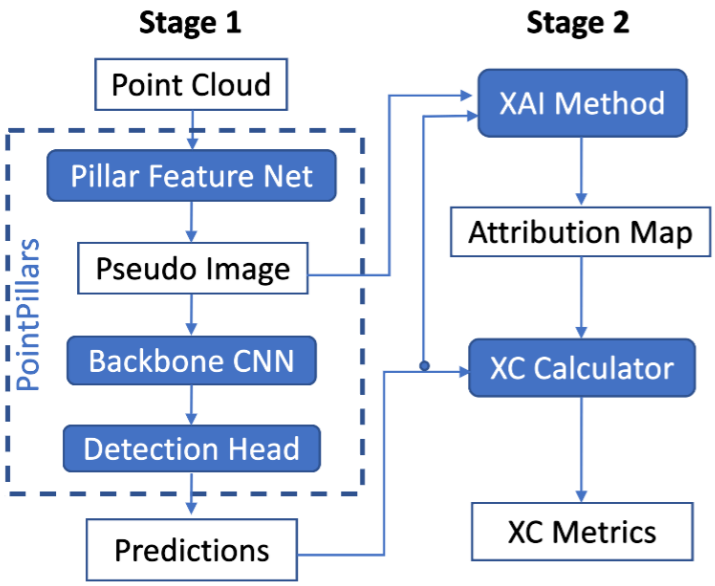


Figure 3.1: Overview of the XC calculation process: PointPillars first process the input point cloud once, then an XAI method computes feature attribution map for specific predictions using the pseudo image as input, and XC metrics are obtained. Blue boxes represent processes, white boxes represent data.

3.1 Explanation in the form of attributions

As mentioned before in Chapter 2, explanation can be presented in the form of input feature attributions, where the magnitude of the attributions corresponds to feature importance, and the sign of attributions indicates positive or negative influence on the model output. In our approach, attributions are generated by IG [53] and backprop [48] for PointPillars [25]. PointPillars has three parts as shown in Figure 3.1: a pillar feature net which learns a 2D bird's eye view (BEV) voxelized feature map called the *psuedo image* from the original point cloud, a backbone 2D CNN to extract more features from the psuedo image, and SSD [32] as detection head to generate 3D object predictions.

IG's effectiveness on image data is well-recognized. However, there are a fixed number of pixels at fixed locations in an image; whereas in a point cloud, both the quantity and location of the points can vary greatly. IG's input transformations may not be meaningful for point cloud. As mentioned before in Section 2.1.4, IG takes a set number of steps from a baseline input x' to the input of interest x. x' is usually set to all zeros. When IG is applied to an image input, the pixel values are the input features to be transformed. The pixels values take steps from 0 to their original values, the locations of the pixels are fixed. Whereas in a 3D point cloud, the points locations themselves are the input features. If one were to apply IG's input transformation to a 3D point cloud directly, then initially all points would be concentrated at (0, 0, 0). Then after each step, they expand outwards a little bit, until eventually the point cloud reaches its original size. Hence, to avoid such undesired shift in input feature locations, we chose to generate attributions using the pseudo image as input, so that IG could be directly applied. Note that our pipeline in Figure 3.1 is not limited to IG or backprop explanations only, any other XAI method that produces explanations in the form of feature attributions could be applied too. Neither is our method limited to the PointPillars model, it can be applied to any object detection model that produces a voxelized representation of the 3D point cloud.

There could be many predictions in the same pseudo image. To explain the different predictions separately, each attribution map is generated for a specific predicted box and its class label. In Figure 3.2, the IG attributions generated for a car prediction is shown. For this particular example, the positive attributions highlight features that increase the model's belief that the object is a car, whereas the negative attributions reduce this belief.

3.2 Motivation for creating the XC scores

We propose a new set of scores called *Explanation Concentration* (XC) which measures the concentration of attributions within a given predicted object's boundaries. As shown in Figure 3.2, many pixels outside of the object have noticeable attributions too, indicating that context features can influence model output as well. The XC scores are partly motivated by previous studies [45, 47] which claimed that overly-relying on context can hurt model performance. Shetty et al. [45] demonstrated that when a 2D image classifier relies too much on the context rather than the object of interest, it can make many mistakes; but model performance is improved when model dependency on context is reduced. Shi et al. [47] also did an ablation study on their PointRCNN model and demonstrated that although small amount of context features can be helpful, too much context information can be detrimental.

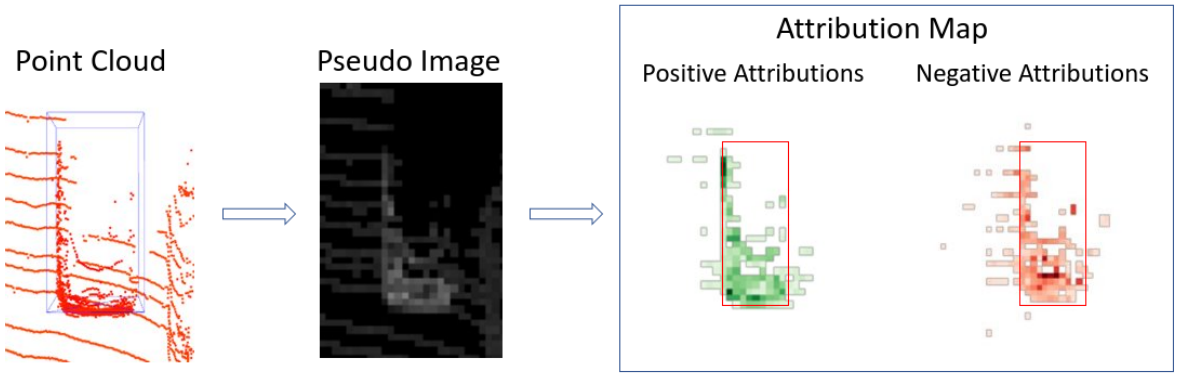


Figure 3.2: Cropped IG attribution map visualization for a car prediction. PointPillars generates a pseudo image from the point cloud, then we create an attribution map for this prediction using the pseudo image as input. Note that the attribution map has the same resolution as the pseudo image. The blue box in the point cloud is the bounding box for the ground truth object. The red box in the attribution map is the 2D projection of the car prediction bounding box. Positive attributions are indicated by green pixels and negative attributions are indicated by red pixels; darker color means greater magnitude. Pixels with attribution values less than 0.1 are filtered out and appear white. The pseudo image and attribution map visualizations are generated by summing values across all channels at each pixel.

To get more information from the pseudo image attribution maps, we then overlay

them on top of the bird's eye view (BEV) of the LiDAR point cloud frame (see Figure 3.3 and Figure 3.4), just like how image attribution maps were overlaid on top of 2D images in previous works. In the point cloud BEV, LiDAR points are shown as white dots and the background empty space is shown in gray. The ground truth bounding boxes are shown as green boxes, the predicted boxes are shown in red, and an additional yellow box is used to highlight the predicted box for which the attributions are generated. As we examine the overlaid attribution maps, we start to discover some trends. Referring to Figure 3.3, one can observe that for true positive (TP) predictions, most of the attributions often resides within the predicted bounding box or right on the box boundary. But referring to Figure 3.4, it is clear that the false positive (FP) predictions often have some pixels with relatively high attribution values outside of the predicted box. These observations lead to the hypothesis that evaluating the concentration of attributions within the predicted box can help identify the prediction as TP or FP.

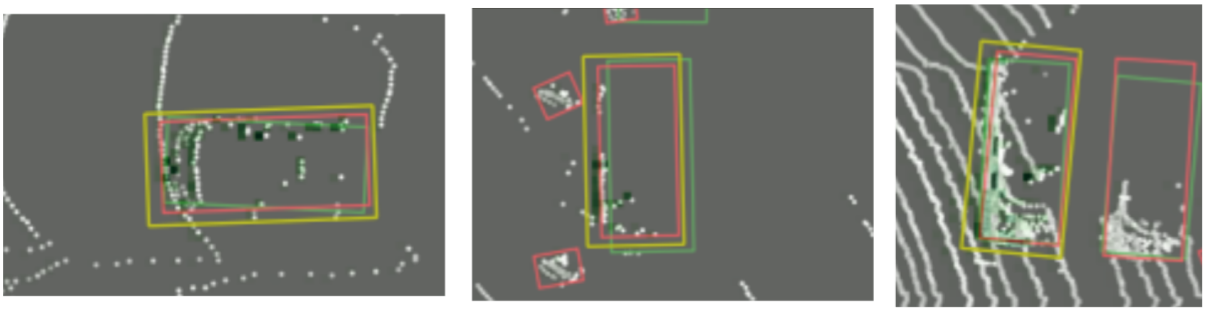


Figure 3.3: Cropped positive IG attribution maps for TP car predictions overlaid on point cloud BEV.

3.3 Computing the XC scores

The process of computing the XC scores is as follows. First, denote the i^{th} predicted box in a pseudo image as $Pred_i$. Then denote the sum of positive attributions across all channels in the pixel at location (x, y) of the attribution map as $a^{+}_{(x,y)}$ (this corresponds to the "Aggregated Positive Attributions" in Figure 3.5). The idea is to avoid having positive and negative channel values cancelling each other out at a specific pixel.

Using thresholds to eliminate noisy signals is a common practice in computer vision. For example, in the Canny edge detection algorithm [3], thresholds are used to mask out weak

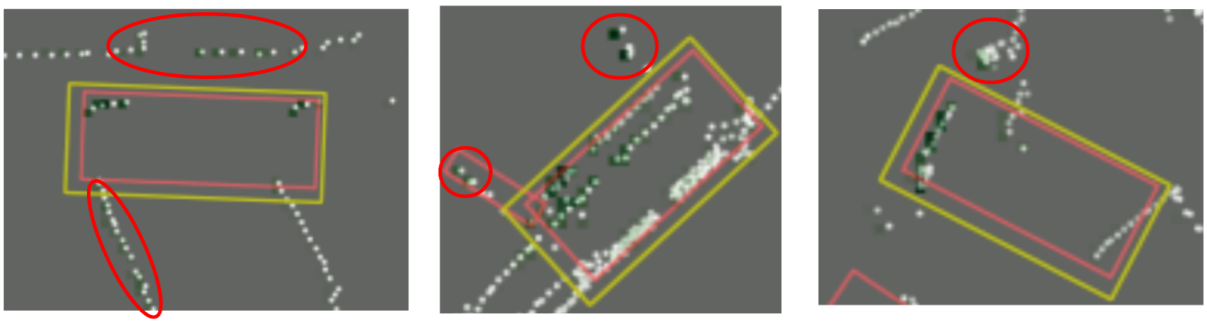


Figure 3.4: Cropped positive attribution maps for FP car predictions overlaid on point cloud BEV. Pixels outside of the predicted box that have high magnitude attributions are highlighted in red circles.

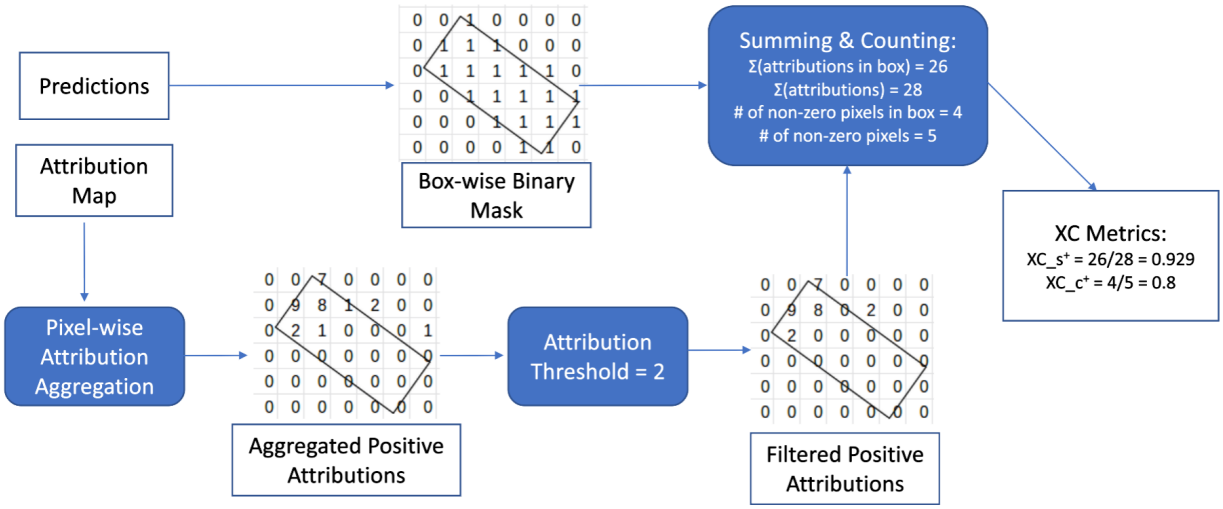


Figure 3.5: The process of computing the XC metrics.

edges. We apply the same idea to the attribution values. Denote a pixel-wise threshold as a_{thresh}, and use it to filter out insignificant attributions (in other words, the sum $a^+_{(x,y)}$ can be called *significant* if it exceeds a_{thresh}). An indicator function $I^+_{(x,y)}$ is defined such that it equals to 1 if $a^+_{(x,y)} >= a_{thresh}$, 0 otherwise. Multiplying the indicator function with the aggregated attributions at each pixel location would produce a filtered attribution map with small attribution values set to zero (this corresponds to the "Filtered Positive Attributions" in Figure 3.5). One way to quantify the concentration of attributions within the predicted box $Pred_i$ is by summing. Two new variables are defined for each $Pred_i$ in a pseudo image:

$$s_i^+ = \sum a^+_{(x,y)} \times I^+_{(x,y)},\ \forall (x,y) \text{ in } Pred_i \tag{3.1}$$

$$S_i^+ = \sum a^+_{(x,y)} \times I^+_{(x,y)},\ \forall (x,y) \text{ in pseudo image} \tag{3.2}$$

Given the box parameters of $Pred_i$, one could generate a box-wise binary mask to indicate if a pixel is located within $Pred_i$ or not (see Figure 3.5), so that s_i^+ could be computed. Now the stage is set for defining an XC score by summing:

$$XC\_s_i^+ = s_i^+ / S_i^+ \tag{3.3}$$

$XC\_s_i^+$ is thus the proportion of the positive attributions which lie within $Pred_i$.

The other way to quantify the concentration of attributions for $Pred_i$ is by counting. For this purpose, we count the number of pixels in the pseudo image having significant attributions, rather than summing them:

$$c_i^+ = \sum I^+_{(x,y)},\ \forall (x,y) \text{ in } Pred_i \tag{3.4}$$

$$C_i^+ = \sum I^+_{(x,y)},\ \forall (x,y) \text{ in pseudoimage} \tag{3.5}$$

$$XC\_c_i^+ = c_i^+ / C_i^+ \tag{3.6}$$

Computing XC by counting might be helpful, because often there are a few outlier pixels with high magnitude attribution values outside of the bounding bounding box (see Figure 3.6 for example). When XC is computed by counting, these outliers will not have a big effect; but when XC is computed by summing, the values of these outlier pixels can skew the resulting XC value towards the lower end. One may also compute a similar set of scores by considering the negative attributions only: call them $XC\_s_i^-$ and $XC\_c_i^-$ for a certain box $Pred_i$.

We observe that pixels located at object boundaries often get labelled as "outside of the box" when creating the box-wise binary mask (see Figure 3.7). Imagine a scenario where

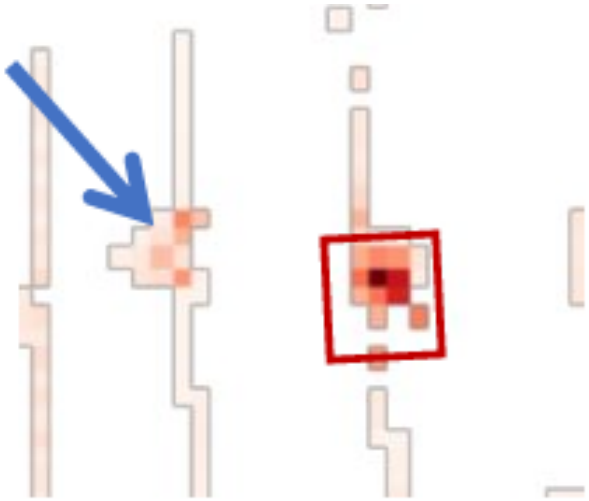


Figure 3.6: Negative attribution map for a TP pedestrian prediction (the red box). The patch of outlier attributions is pointed out by the arrow. To highlight just the attributions, the point cloud BEV and ground truth bounding box are not shown.

most of the attributions assigned to a prediction are located on the object boundary. This prediction may have very low XC due to the fact that pixels on the boundary are not considered as belonging to the predicted object. But this is unreasonable, as most of the attributions are indeed assigned to pixels on the predicted object and the XC value should be high. Hence, before calculating any XC scores, the predicted boxes are enlarged by a small margin m on all sides, so that pixels at object boundaries are labeled as inside the predicted box.

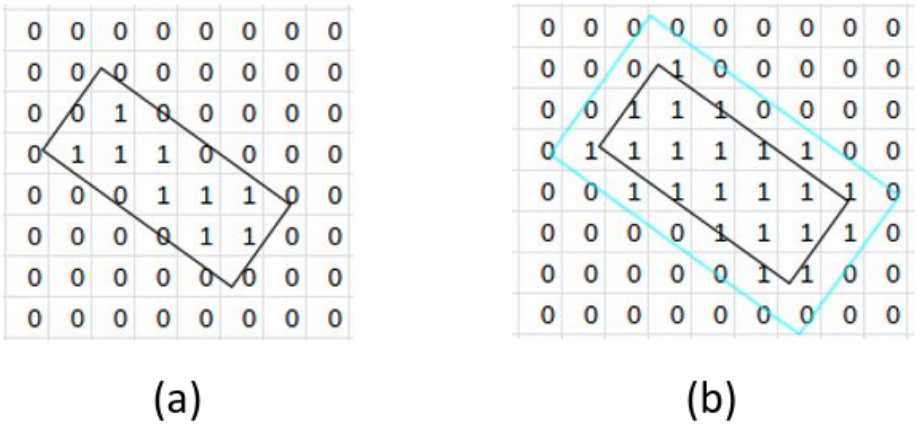


Figure 3.7: The effect of padding on the box-wise binary mask: a) In the binary mask generated from the original bounding box (black box), a lot of the pixels on the object boundary get labelled as "0" (outside of the object) rather than "1" (inside the object); b) in the binary mask generated from an enlarged bounding box (the blue box), the pixels on the original object boundary (the black box) are now labelled as "1". Note that the pixel size and predicted box size are not drawn to scale.

Chapter 4

Using XC to help distinguish TP vs. FP predictions

4.1 Evaluation metrics and implementation details

The objective of our experiment is to evaluate XC's performance on a meta classification task: classifying predictions as either FP or TP. The predicted objects are categorized as TP or FP as illustrated in Figure 4.1. In essence, a TP prediction has a matching ground truth object but a FP prediction does not. We will first explain the scoring mechanism of the model we use, then we explain how we decide if a prediction is TP or FP. In OpenPCDet's [54] implementation of PointPillars [25], the class scores are not softmax scores. Rather, they are class-wise sigmoid scores: a pair of "class vs. not class" scores for each object class. For example, a predicted box would have a pair of sigmoid scores for "car vs. not car" probabilities, a pair for "pedestrian vs. not pedestrian", and another pair for "cyclist vs. not cyclist", each pair sums up to 1. The "not" probabilities are discarded, and the "car", "pedestrian", "cyclist" probabilities are retained, with the highest one determining the class label.

For one particular frame, any predictions with the highest class score less than a threshold S_{th} is ignored. The IoU between the remaining predictions and the ground truth (GT) objects in the frame are computed. In order for a prediction to be labelled as TP, its maximum IoU with any GT objects needs to exceed a threshold IoU_{th} and its label needs to match with the label of the GT object that has the highest IoU with it, otherwise it's considered as FP (as shown in Figure 4.1). Based on KITTI's [14] conventions, S_{th} is set to

be 0.1 for all predictions, and IoU_{th} values are 0.5, 0.25, and 0.25 for 3D IoU for predictions labeled as car, pedestrian, and cyclist respectively.

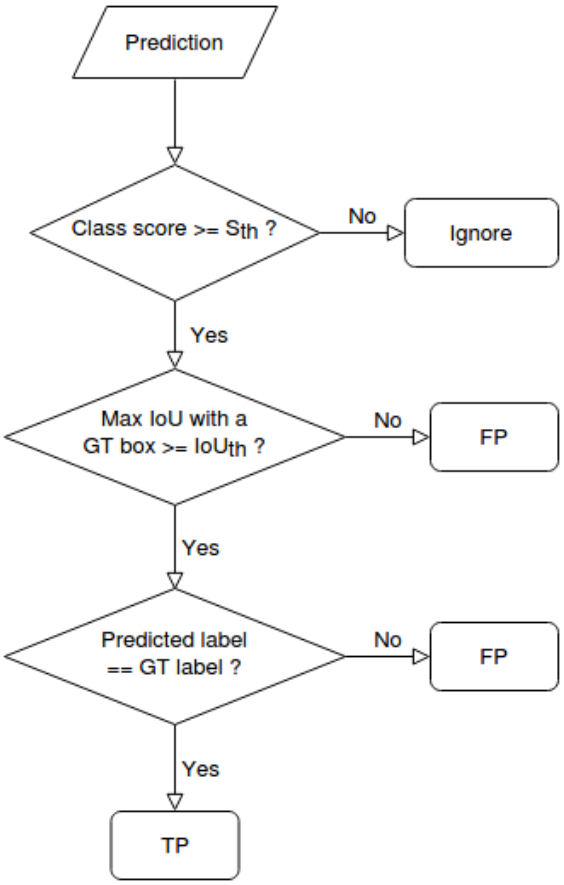


Figure 4.1: The process of labeling a prediction as either TP or FP.

To evaluate the performance of a specific score, we could simply apply a score threshold: if the score is above the threshold then the corresponding prediction is TP, otherwise it's FP. Then we can evaluate the resulting detection accuracy. However, the resulting accuracy is a function of the threshold. To remove the effect of threshold selection and evaluate the performance of different XC scores more fairly, we compute area under the precision recall curve (AUPR) [35] and area under the receiver operating characteristics curve (AUROC) [8], both are threshold-independent performance measures for binary classification.

In binary classification tasks, typically one class is treated as the "positive" class,

whereas the other class is treated as the "negative" class. We may choose to call the TP predicted boxes "positive" and call the FP predicted boxes "negative" or vice versa. We can then apply a threshold to an XC score to classify the predicted boxes as positive or negative (e.g., say $>= 0.5$ is positive, otherwise negative).

Referring to the confusion matrix presented in Table 2.1, one can then compute the precision (TP/(TP+FP)) and recall (TP/(TP+FN)) values for that particular score threshold. Similarly, true positive rate (TPR = recall) and false positive rate (FPR = FP/(FP+TN)) can also be calculated. One can then repeat the same process across the range of possible score thresholds to obtain many pairs of precision vs. recall and TPR vs. FPR values to produce the PR curve and ROC curve respectively. Hence the reason why AUPR and AUROC are threshold-independent performance measures. Note that in this paragraph, in the formulae for precision, recall, TPR, and FPR, TP and FP are defined in the context of binary classification, they are not to be confused with the TP and FP predicted objects in the context of object detection (which in this case are the instances to be classified).

The AUROC metric treats both classes equally and can reflect the score's (e.g., one of the XC scores) ability in correctly identifying both the positive and negative classes. On the other hand, the AUPR metric puts more emphasis on the score's ability to correctly identify the positive class. For both AURP and AUROC, higher values indicate better performance.

We evaluate the XC scores on three PointPillars [25] models trained on the KITTI dataset [14] and on one PointPillars model trained on the Waymo dataset [52]. We use OpenPCDet's [54] implementation of PointPillars for our experiments and adapt their default settings for training. The first three models are trained for 80 epochs on the KITTI dataset, with 3712 frames for training and 3769 frames for validation. Due to time and resource constraints, we obtain only one more model trained for 30 epochs on 20% of the Waymo dataset, with 31616 frames for training and 7997 frames for validation. To obtain attribution values, we use Captum [50], a model interpretability library developed for PyTorch [36]. We explore both IG [53] and backprop [48] as explanation methods. For IG, we choose to use 24 steps to obtain the Riemann sum approximation using the trapezoidal rule in order to save time while retaining some accuracy (since the authors suggested 20 to 300 steps).

For KITTI, the XC scores are obtained on the predicted objects from the 3769 validation frames. There are on average 67k predicted objects produced by each model. Each pixel in the pseudo image has 64 channels, and the size of the pseudo image is $496 \times 432 \times 64$, representing features learned in a cropped point cloud of size $79.36m \times 69.12m \times 4.00m$ $(width \times length \times height)$. This means that one pixel in the pseudo image encodes

Table 4.1: The effect of a_{thresh} on XC_c^+'s ability in classify TP vs. FP predictions. Results are gathered from predictions generated from the KITTI validation set. The best value for each metric is highlighted in bold. Note that AUPR_op is the AUPR obtained by treating the FP predictions as positive instances.

a_{thresh}	AUROC	AUPR	AUPR_{op}
0.0	0.7162	0.3552	0.8897
0.0333	0.8351	0.6321	0.9325
0.0667	0.8569	0.6671	0.9413
0.1	**0.8634**	0.6676	0.9446
0.1333	0.8625	**0.6701**	**0.9452**
0.1667	0.8551	0.6678	0.9434
0.2	0.8448	0.6683	0.9407

point features in a 0.16m ×0.16m ×4.00m pillar. For Waymo, we comptue XC scores for 94k predicted boxes sampled from 800 validation frames. Again, each pixel in the pseudo image has 64 channels, but the size of the pseudo image is $512 \times 512 \times 64$, representing features learned in a cropped point cloud of size 168.96m ×168.96m ×6.00m, ($width \times length \times height$). One pixel in the pseudo image then represents point features in a 0.33m ×0.33m ×6.00m pillar. For both datasets, we append a margin $m = 0.2$m to the predicted boxes and apply $a_{thresh} = 0.1$ to the attribution maps. $m = 0.2$m is intentionally selected to be roughly equal to the pseudo image pixel size to prevent the edge effect described in Figure 3.7. $a_{thresh} = 0.1$ is selected through an experiment on XC_c^+ applied to predictions from one of the PointPillars models trained on KITTI. As shown in Table 4.1, when a_{thresh} is 0.1 or 0.1333, the performance of XC_c^+ is relatively good. But as a_{thresh} decreases below 0.1 or increases beyond 0.1333, the XC score's performance degrades.

4.2 Distribution of XC values

To get some confidence on the XC score's ability to distinguish the TP and FP predictions, we choose to plot the distribution of XC_c^+ values obtained from one of the models trained on KITTI [14] (shown in Figure 4.2). Both the histograms and the empirical cumulative distribution function plots demonstrate that the XC values of TP instances tend to be greater than those of the FP instances. Another notable observation is that XC values

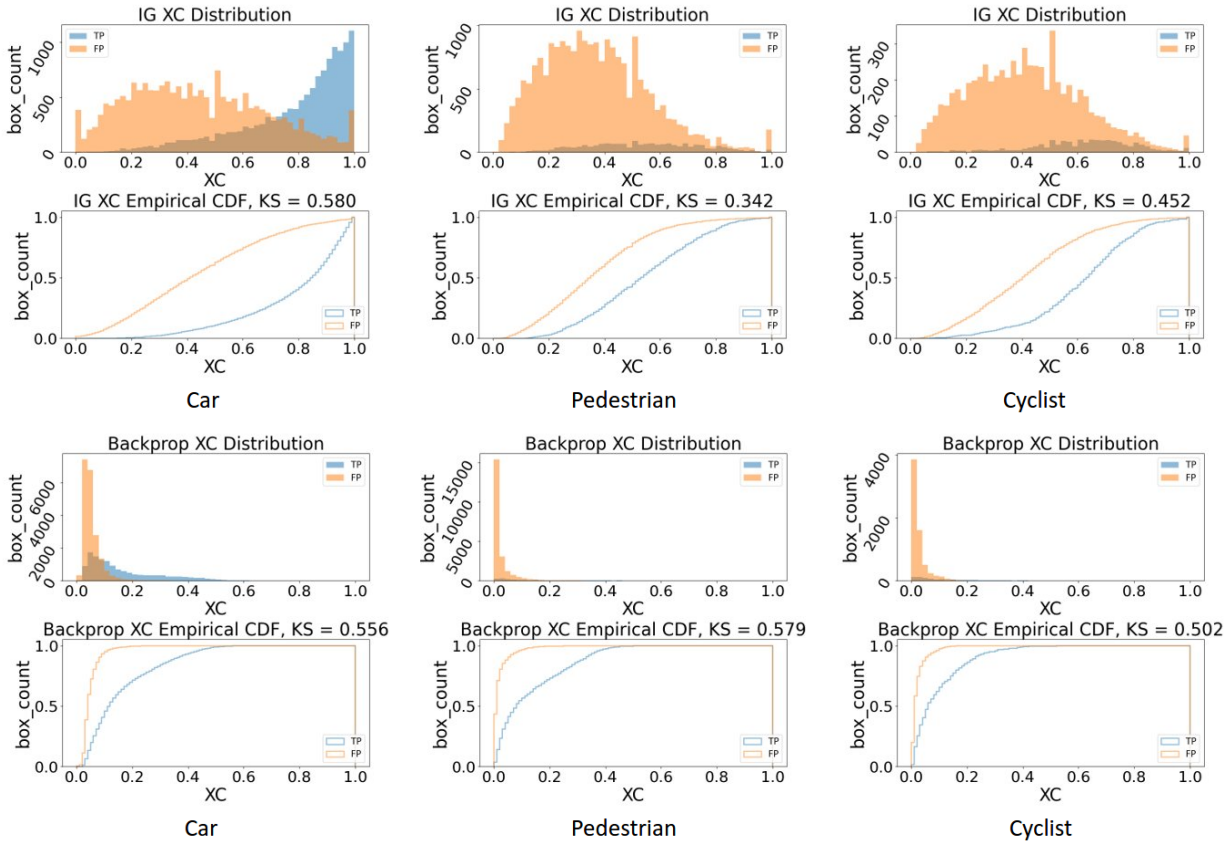


Figure 4.2: Distribution of XC_c^+ values obtained by IG (upper row) and backprop (lower row) attributions for TP and FP predicted boxes in each of the object class of the KITTI validation set. "KS" means the Kolmogorov-Smirnov statistic [9].

derived from IG are well-dispersed within the range of $[0, 1]$ for both the TP and FP instances, whereas XC values derived from backprop are mostly below 0.5, with almost all FP instances having XC values below 0.2.

4.3 XC's relationship with other box-wise features

To understand the correlation between XC and other box-wise features, we pick one of the three PiontPillars models trained on KITTI [14] to obtain predictions on the KITTI validation set and plot XC_c^+ value for each prediction with other box-wise features in Figure 4.3, Figure 4.4, and Figure 4.5. The predictions with different object class labels are plotted separately. Each dot represents a prediction. For example, say in one of the plots in Figure 4.3, there is a dot at coordinate (0.7, 0.3), then this dot represents a prediction with top class score equals to 0.7 and XC_c^+ equals to 0.3. The color in these plots indicates relative density: red highlights relatively densely populated regions, whereas dark blue highlights relatively sparse regions.

It is important to examine the correlation between XC and other box-wise features before further experimentation. When two features have very strong correlation, then the data points would all cluster very closely along a line in a 2D plot, with each axis representing each of the two feature values. In this case, one would not be able to get extra benefits by using both of these features. Since they are so strongly correlated, knowing one feature value would be sufficient to derive the other feature value. Hence, for an XC score to be able to provide additional value, they should not demonstrate strong correlation with any other box-wise features. Referring to Figure 4.3, Figure 4.4, and Figure 4.5, it is clear that XC_c^+ does not demonstrate strong correlation with top class score, distance to LiDAR sensor, or number of points in predicted box for any of the three object classes of interest. We are now confident that the XC scores will be able to reveal new insights about the predictions.

There are some additional observations worth-mentioning. The separation between the high density regions of the TP and FP predictions is often the greatest in the car predictions, but less in the pedestrian and cyclist classes. Hence when using these features to distinguish TP and FP predictions, the performance on the car predictions is likely to be better than that on the other two classes (see Section 4.4 for more details on related results). The separation between the TP and FP clusters is also the most obvious when the top class score and XC_c^+ form the 2 axes of the plots. This observation indicates that combining the XC scores with top class score might be more beneficial for the task of classifying TP vs. FP predictions compared to combining the XC scores with number

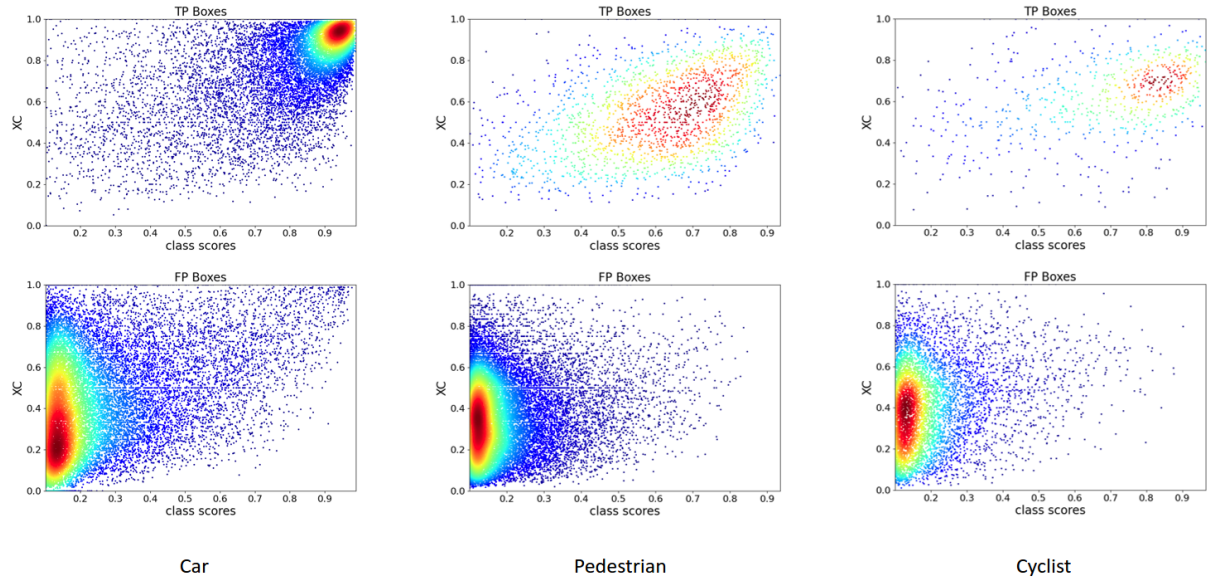


Figure 4.3: XC_c^+ value and top class score for predicted boxes in KITTI validation set.

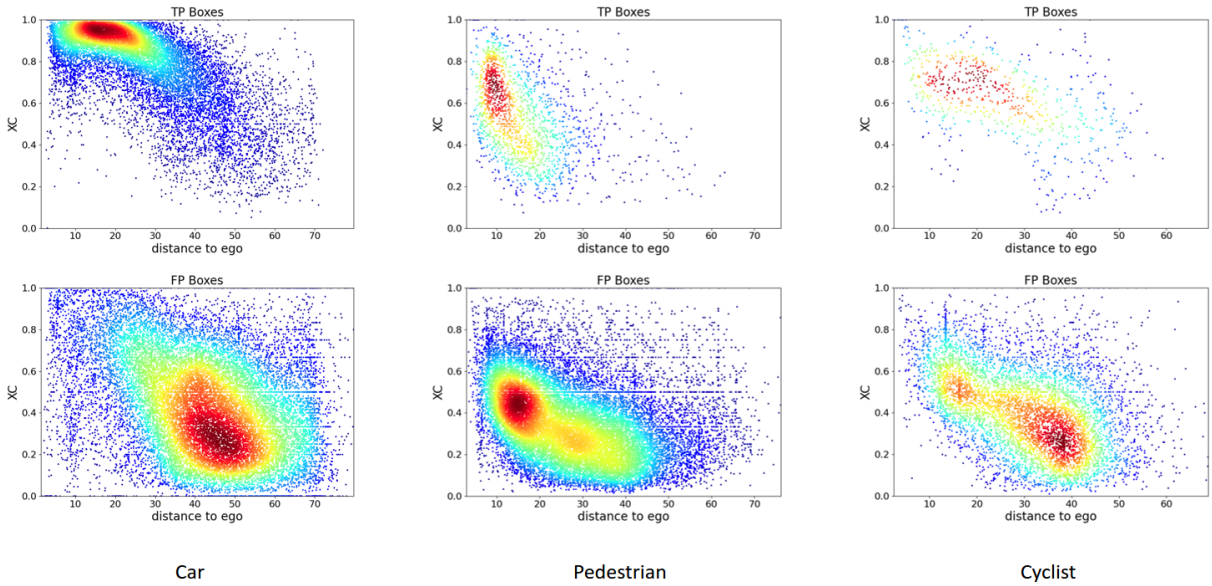


Figure 4.4: XC_c^+ value and distance to LiDAR sensor for predicted boxes in KITTI validation set.

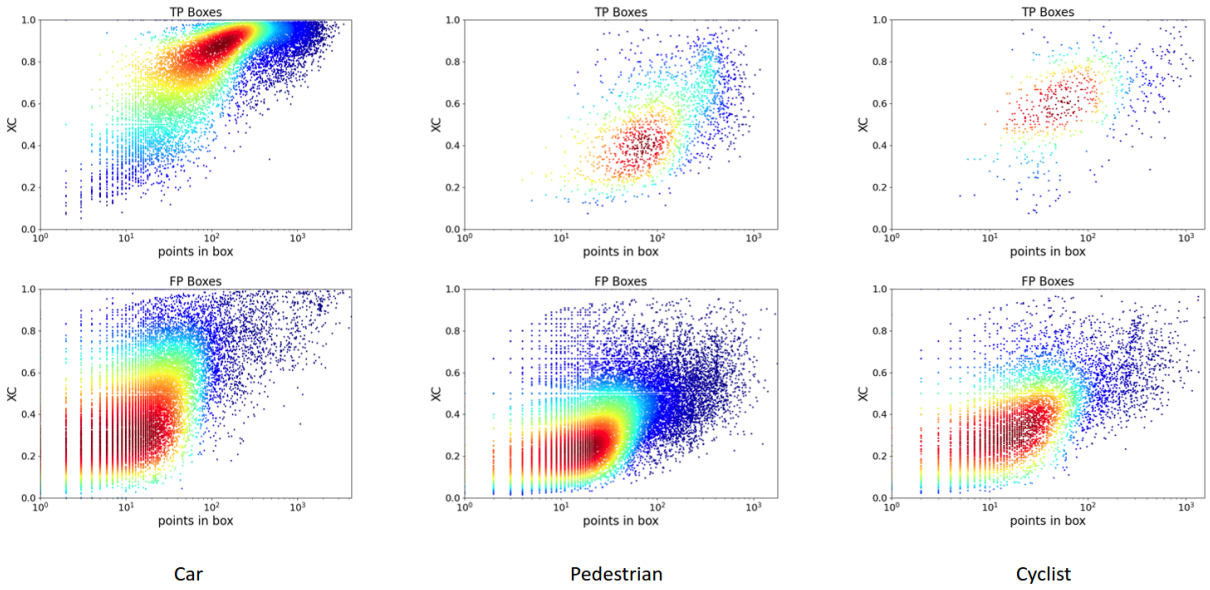


Figure 4.5: XC_c^+ value and number of LiDAR points within bounding box for predicted boxes in KITTI validation set. Note that the number of LiDAR points is plotted in log scale due to the fact that most bounding boxes contain less than 100 points.

of points or distance to LiDAR sensor. Hence we tried to further boost performance by combining the XC scores with top class score in Section 4.6.

4.4 Using only XC to identify TP and FP predictions

The results for TP vs. FP box classification using the XC scores on the KITTI dataset [14] are shown in Table 4.2. Each specific metric in this table is averaged over three models. The four XC scores derived from IG and backprop are evaluated, along with four other box-wise features (random guess, distance of the predicted box to LiDAR sensor, the number of points inside the predicted box, and the highest class score for the predicted box) serving as baselines for comparison. Each of the aforementioned features are evaluated by three metrics on different object classes, making up twelve metrics in total. Note that the object classes in Table 4.2 (as well as those in other tables in later sections) represent the predicted labels, not the ground truth labels.

When computing AUROC and AUPR, the TP boxes are treated as the positive instances, but AUPR_op refers to the AUPR value obtained by treating the FP predicted boxes as positive instances. In essence, AUPR reflects TP detection performance, whereas AUPR_op reflects FP detection performance. The expected AUROC value for a random score is 0.5, and the expected AUPR value is the proportion of the positive instances.

Referring to Table 4.2, it is clear that the top class score beats all other features in every evaluation metric. However, the improvement brought by XC is certainly non-trivial. One can observe that for all features evaluated, the second best performing feature is most often $XC\_c_B^-$. Although distance to sensor and number of points in predicted box beat the XC scores generated by IG on the three pedestrian class metrics, they are unable to beat any of the XC scores generated by backprop on the pedestrian class.

A very notable case is the AUPR for pedestrian predictions: $XC\_c_B^-$ achieves 129% improvement compared to number of points and 706% improvement compare to random guess. The other three XC scores derived from backprop also achieve more than 100% on AUPR for pedestrian compared to the number of points. Improvements of similar magnitude are also achieved by the backprop XC scores on AUPR for the cyclist class. These observations indicate that the backprop XC scores are much better at correctly identifying the TP predictions for the pedestrian and cyclist classes than simple heuristics such as number of points. Another interesting observation is that the XC scores derived by counting usually outperforms those derived by summing. Such improvement might be due to the outlier attenuation effect mentioned in the second last paragraph of Section 3.3.

Table 4.2: Comparison of the XC scores' ability to classify TP and FP predictions for different object types in the KITTI dataset. The subscripts "IG" and "B" indicate whether the corresponding XC score is derived from IG or backprop attributions. For each evaluation metric, the XC scores performing worse than number of points are highlighted by underscore and the best performing feature other than the top class score is highlighted in **bold**.

Metrics	Random	Distance	Points	$XC\_s^+_{IG}$	$XC\_c^+_{IG}$	$XC\_s^-_{IG}$	$XC\_c^-_{IG}$	$XC\_s^+_{B}$	$XC\_c^+_{B}$	$XC\_s^-_{B}$	$XC\_c^-_{B}$	Top Class Score
AUROC												
All	0.5	0.669	0.720	0.843	0.868	0.827	0.869	0.823	0.903	0.822	**0.908**	0.971
Car	0.5	0.770	0.779	0.837	0.857	0.822	0.857	0.708	0.861	0.698	**0.869**	0.964
Pedestrian	0.5	0.779	0.832	0.648	0.699	0.671	0.754	0.864	0.888	0.882	**0.894**	0.958
Cyclist	0.5	0.635	0.780	0.810	0.797	0.806	0.824	0.808	0.843	0.819	**0.855**	0.965
AUPR												
All	0.232	0.372	0.464	0.585	0.653	0.551	0.635	0.597	0.786	0.577	**0.794**	0.926
Car	0.391	0.636	0.666	0.713	0.749	0.697	0.747	0.621	0.830	0.597	**0.836**	0.948
Pedestrian	0.071	0.215	0.250	0.116	0.153	0.136	0.190	0.518	0.557	0.541	**0.572**	0.759
Cyclist	0.083	0.167	0.203	0.225	0.237	0.239	0.270	0.425	0.534	0.450	**0.554**	0.829
AUPR_op												
All	0.768	0.859	0.878	0.941	0.950	0.937	0.952	0.936	0.965	0.937	**0.967**	0.989
Car	0.609	0.829	0.839	0.894	0.903	0.885	**0.905**	0.761	0.888	0.761	0.897	0.974
Pedestrian	0.929	0.976	0.984	0.959	0.967	0.962	0.973	0.986	0.989	0.988	**0.990**	0.996
Cyclist	0.917	0.939	0.976	0.977	0.975	0.978	0.980	0.975	0.979	0.977	**0.982**	0.992

Table 4.3: Comparison of the XC scores' ability to classify TP and FP predictions for different object types in the Waymo dataset. For each evaluation metric, the XC scores performing worse than number of points are highlighted by underscore and the best performing feature other than the top class score is highlighted in **bold**.

Metrics	Random	Distance	Points	$XC\_s^+_{IG}$	$XC\_c^+_{IG}$	$XC\_s^-_{IG}$	$XC\_c^-_{IG}$	$XC\_s^+_{B}$	$XC\_c^+_{B}$	$XC\_s^-_{B}$	$XC\_c^-_{B}$	Top Class Score
AUROC												
All	0.5	0.609	0.701	0.714	0.758	0.738	0.766	0.729	0.788	0.721	**0.793**	0.965
Vehicle	0.5	0.703	0.809	0.799	0.821	0.806	0.823	0.754	**0.860**	0.725	**0.860**	0.982
Pedestrian	0.5	0.528	0.614	0.529	0.529	0.529	0.545	0.605	0.638	0.646	**0.651**	0.927
Cyclist	0.5	0.627	0.738	0.766	0.725	0.673	0.689	0.794	0.823	0.799	**0.823**	0.979
AUPR												
All	0.276	0.343	0.476	0.460	0.535	0.483	0.540	0.569	**0.700**	0.542	0.699	0.936
Vehicle	0.377	0.595	0.721	0.626	0.659	0.624	0.654	0.678	**0.827**	0.648	0.824	0.973
Pedestrian	0.176	0.121	0.147	0.183	0.184	0.181	0.192	0.284	0.323	0.326	**0.340**	0.829
Cyclist	0.051	0.072	0.113	0.111	0.096	0.078	0.082	0.279	0.377	0.283	**0.394**	0.791
AUPR_op												
All	0.724	0.811	0.862	0.863	0.881	0.874	**0.884**	0.850	0.876	0.857	0.882	0.983
Vehicle	0.623	0.784	0.873	0.878	0.891	0.883	0.892	0.809	0.891	0.786	**0.893**	0.987
Pedestrian	0.824	0.897	**0.925**	0.841	0.839	0.842	0.846	0.866	0.882	0.889	0.888	0.979
Cyclist	0.950	0.967	0.984	0.984	0.980	0.976	0.978	0.984	0.986	0.985	**0.986**	0.999

To ensure that the advantages offered by XC are not specific to the KITTI dataset [14], we also present results from Waymo dataset [52] in Table 4.3. Again, for most evaluation metrics, one of the XC scores is the second best performing feature besides the top class score. In addition, the XC scores obtained from backprop often show 100% or more improvement on AUPR for pedestrian and cyclist predictions compared to the number of points. And again, $XC\_c_B^-$ is the best performing feature in most cases. Hence, we believe that the advantages of XC are not limited to the KITTI dataset only.

4.5 Why backprop outperforms IG?

IG [53] is designed to reflect feature importance more precisely than other simpler XAI methods such as backprop. Thus, it would be interesting to know why IG-based XC scores underperform backprop-based XC scores in the task of classifying TP vs. FP predictions, especially for the pedestrian and cyclist predictions (see AUPR in Table 4.2 and Table 4.3). We suspect that this is due to the difference in XC distribution. In Figure 4.2, we present the KS statistic [9] between the distributions of XC values in the TP and FP instances of each object class. The KS statistic is a measure for goodness of fit between two distributions: greater value indicates greater difference between the two distributions. Note that the backprop-based XC scores are able to produce much KS statistic between TP and FP distributions in the pedestrian and cyclist class than the IG-based scores.

To alter the distribution of IG-based XC scores, we remove the last step in computing IG attributions. As mentioned before in Chapter 2, IG zeros out attributions for input features with zero value. This is often achieved by multiplying the computed attributions at each input location i by $(x_i - x_i')$, where x is the input and x' is the baseline, which is by default set to zero. We remove this process and re-calculate IG-derived XC scores on the KITTI dataset [14]. Then we evaluate the new XC scores on the binary classification task for TP vs. FP predictions. The average results of all four IG-derived XC scores, all four backprop-derived XC scores (these are averaged over the values presented in Table 4.2), and of the four modified IG-derived XC scores are presented in Table 4.4. Note that the results are also averaged over all three models trained on KITTI. The modified IG XC scores resulted in 115% improvement in AUPR for pedestrian predictions and in 16% improvement in AUPR for cyclist predictions compared to the original IG XC scores. Improvement can also be observed in AUROC and AUPR_op for the pedestrian and cyclist classes.

The improvements on pedestrian and cyclist objects also coincide with a shift in the distribution of XC values. As shown in Figure 4.6, the distribution of XC values now appears very similar to that of backprop-based XC values, but very different from that

Table 4.4: Average XC performance on distinguishing TP vs. FP predictions for the KITTI dataset. The highest value in each row is highlighted in **bold**.

Metrics	XC_{IG}	Modified XC_{IG}	XC_B
AUROC			
All	0.852	0.861	**0.864**
Vehicle	**0.843**	0.795	0.784
Pedestrian	0.693	0.848	**0.882**
Cyclist	0.809	0.812	**0.831**
AUPR			
All	0.606	0.657	**0.689**
Vehicle	**0.726**	0.707	0.721
Pedestrian	0.149	0.321	**0.547**
Cyclist	0.243	0.282	**0.490**
AUPR_op			
All	0.945	**0.952**	0.951
Vehicle	**0.897**	0.848	0.827
Pedestrian	0.965	0.986	**0.988**
Cyclist	0.977	**0.978**	**0.978**

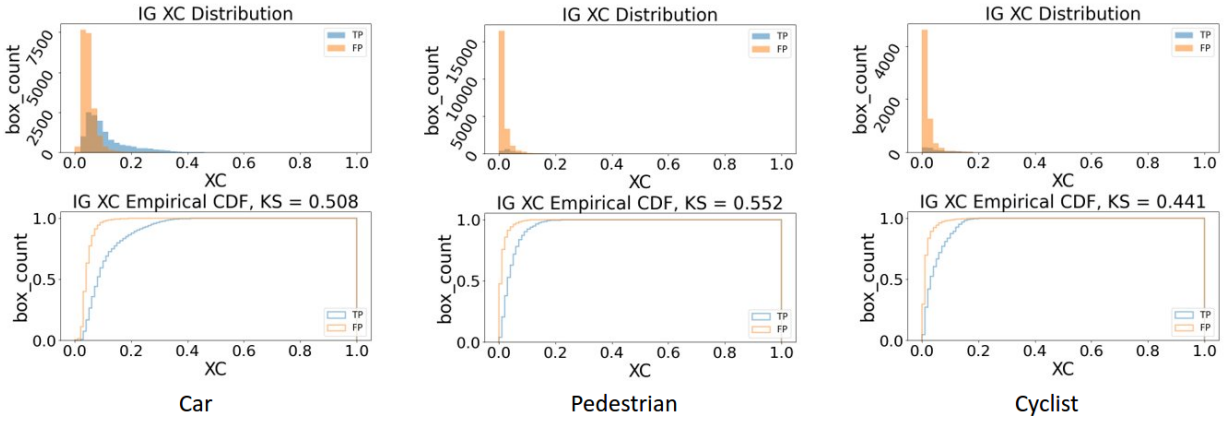


Figure 4.6: Distribution of XC_c^+ values obtained by modified IG (without multiplying attributions by input) attributions for TP and FP predicted boxes in the KITTI validation set.

of IG-based XC values. By making feature attributions proportional to input feature magnitude, IG is able to generate attribution maps that capture salient features in the input, which is one reason why IG attribution maps makes more sense to a human observer compared to a blurry backprop attribution map (interested readers may visit Sundararajan et al. [53] for more attribution map examples). As a result, IG zeros out most of the attributions outside of the bounding box (because the space outside of object bounding boxes are mostly empty, leading to zero input feature values at such locations), and the remaining attributions are mostly concentrated within the bounding box or at its close proximity (see Figure 3.2). Thus, IG is unable to produce mostly very low (¡ 0.1) XC values for the FP predictions, leading to significant overlap in the values for TP prediction XC scores and FP predictions XC scores, making classification using XC scores difficult.

These observations echo with Erion et al.'s [11] claim that IG's choice of a zero-valued baseline is problematic. Take an image of a digit for example, if the background is white but the digit itself is black (i.e., zero), then the zero-valued pixels in fact contains the key features of this image and should not get zero attributions. Similarly for point cloud inputs, not just the presence of points, but also the absence of points in certain locations can help the model classify the object. For instance, pedestrian objects are usually filled with points, whereas car objects have points on its boundaries but are mostly hollow in the middle.

Note that even without multiplying by input values, IG-derived XC scores still cannot beat backprop-derived XC scores on the pedestrian and cyclist classes (see Table 4.4). Hence, a more expensive method such as IG may produce more visually appealing attribution map, but a less expensive method may have more potential when used quantitatively.

4.6 Combining XC scores with the top class score

In Section 4.4 we demonstrate that the top class score is the best performing box-wise feature in classifying TP vs. FP predictions. In this section, we aim to improve its performance by combining it with XC scores generated from backprop attributions. We train classifiers for different object classes in KITTI [14] and perform an ablation study on the box-wise features.

To conduct the ablation study, we build a new dataset D_f from five features (the top classs score and the four XC scores computed based on backprop attributions) for each predicted box produced by one PointPillars [25] model on the KITTI validation set [14]. Since we have three models trained on KITTI, we obtain three D_f datasets. For each D_f,

we first group the samples by the predicted object label, then by the number of LiDAR points: those with less than 100 points forms one set, the rest forms another set. Thus, from one D_f, we generate six smaller datasets d_f, two for each of the three object classes.

We then build a 2-layer multilayer perceptron (MLP) with PyTorch [36] as a classifier to be trained on d_f. Layer 1 is of size $(d \times 3)$ and layer 2 is of size (3×1), where d represents the number of input features per instance. ReLU activation is applied after the first layer, and the sigmoid function is applied after the second layer to obtain an output score. We train the MLP to classify a predicted box as TP or FP, using binary cross entropy as the loss function. All input features are normalized based on the following equation prior to being fed into the MLP: $z = (x - \mu)/s$, where x is the original feature value, μ is feature mean value, and s is the standard deviation for that feature. We use the Adam optimizer [22] with learning rate set to 0.001.

For each experiment, we first shuffle d_f and then augment it by duplicating the instances four times. Next we apply 5-fold cross validation to the augmented d_f, obtaining 5 different 80%/20% train/validation splits. For the training instances, we also add a small uniformly distributed noise $U(-0.05, +0.05)$ to each feature to help the MLP generalize better. For each different split, we train the MLP for 12 epochs with batch size = 16 and record three evaluation metrics (AUROC, AUPR, and AUPR_op) of the output score on the validation instances. We repeat the 5-fold cross validation 5 times, obtaining $5 \times 5 = 25$ different values for each of the metrics, and record the average. Note that the above process is repeated for each d_f in the three D_f we have, each evaluation metric is averaged over the three D_f and shown in Table 4.5.

In Table 4.5, when the top class score is the only input feature, we evaluate its performance directly and use it as baseline for comparison; when more than one features are used, we evaluate the performance of the MLP output score. The most notable observation is that for predictions containing less than 100 points, combining the XC scores with top class score can often result in better performance in distinguishing TP vs. FP predictions, especially among the pedestrian predictions. The AUPR for pedestrian increased by (0.540 - 0.492) / 0.492 = 9.8% after combining the 4 XC scores with top class score. The improvement in AUROC for pedestrian predictions and in AUPR for cyclist predictions also exceed 1%. Among the predictions with more than 100 points, the benefit of incorporating the XC scores is less observable. Note that the top class score alone is already performing very well for these predictions with more points. This might be why it is more difficult to get additional benefit from the XC scores on these predictions.

Table 4.5: Ablation study on the features used to help classify TP vs. FP predictions on KITTI.

Object Class	Features Used					Points < 100			Points >= 100		
	Top class score	XC_c^-	XC_c^+	XC_s^-	XC_s^+	AUROC	AUPR	AUPR_op	AUROC	AUPR	AUPR_op
Car	✓					0.958	0.926	0.977	0.956	0.980	0.914
	✓	✓				0.958	0.927	0.977	0.950	0.978	0.903
	✓	✓	✓	✓	✓	0.960	0.927	0.979	0.957	0.980	0.918
Pedestrian	✓					0.932	0.492	0.997	0.969	0.911	0.989
	✓	✓				0.938	0.527	0.997	0.973	0.919	0.991
	✓	✓	✓	✓	✓	0.944	0.540	0.997	0.973	0.920	0.992
Cyclist	✓					0.958	0.765	0.996	0.982	0.947	0.995
	✓	✓				0.958	0.777	0.996	0.972	0.931	0.993
	✓	✓	✓	✓	✓	0.958	0.779	0.996	0.980	0.946	0.994

Chapter 5

Using quantitative measures of explanations to train object detector

5.1 Designing loss functions based on attributions

5.1.1 Loss formulations

We have shown that the XC scores can help identify TP vs. FP predictions. We now wish to know if we can use XC or pixel attribution prior (PAP) [11] to improve model performance. Inspired by Ross *et al.*'s work [43], we design our own loss terms based on XC or PAP.

Before diving into the loss terms themselves, it is important to discuss the high computational cost of training with attribution-based losses. An XC score is a function of the attribution map, which is the same size as the pseudo image for PointPillars [25] in our case. We can now roughly estimate how many extra parameters would be included in the computational graph if we apply a loss based on the attribution map. In the case of backprop, the attribution map is the gradient of the model output with respect to the pseudo image features. To compute the attribution map, one would need to compute all partial derivatives $\frac{\partial f}{\partial \theta}$ from the output layer all the way back to the pseudo image using chain rule, where f is the model output we wish to explain (class score for the predicted label in our case) and θ can be any parameter in the pseudo image. There is a 2D backbone CNN and a SSD network [32] between the pseudo image and the output (see Figure 3.1). Hence, the attribution map is a function of all the parameters in the 2D CNN and SSD.

We can denote their sizes as $|\Theta_{CNN}|$ and $|\Theta_{SSD}|$. Any loss based on the attribution map would eventually be backpropagated into the CNN and the SSD.

Now consider that the XC score is a function of all the values in the attribution map and of the predicted bounding box parameters. We need to compute $\frac{\partial XC}{\partial \phi}$ for all values ϕ in the attribution map to just backpropagate a loss based on XC into the attribution map. Therefore, creating a loss term based on an XC score for only one prediction in a LiDAR frame would make the computational graph at least this much larger:

$$|\Theta_{CNN}| + |\Theta_{SSD}| + |\Phi| \tag{5.1}$$

where Φ is the attribution map. In our experiments on KITTI, the attribution map has about 14 million parameters. Therefore, it is easy to see that incorporating a loss based on the attribution values would significantly slow down the training process, even for just one prediction per frame. In addition, just computing the attributions and XC values would be a notable overhead too, especially for IG. It is unrealistic to compute XC score for all predictions during every epoch of training.

Taking all these above-mentioned factors into account, we are going to: 1) limit the number of predicted boxes in each frame for which XC is computed and XC related losses are applied, and 2) conduct most of experiments using attributions based on backprop rather than IG.

We design three loss terms with different objectives: to increase XC values, to reduce XC values, and to reduce pixel attribution prior (PAP) [11]. These loss terms are used in conjunction with the original regression and classification loss terms. The following loss term Ψ aims to increase XC values:

$$\psi_i = \sum_j (1 - XC_j),\ \forall j \in P \tag{5.2}$$

$$\Psi = \lambda_\Psi * \frac{1}{|B|} \sum_i \psi_i,\ \forall i \in B \tag{5.3}$$

We denote the predicted box index as j and the LiDAR frame id as i. XC_j is an XC score computed for a specific prediction j, it can be any of the four XC scores described in Section 3.3. It is clear that as XC approaches one (i.e., its maximum value), the loss would approach zero. P is the set of selected predictions for a frame. To allow reasonable training speed, $|P|$ needs to be much less than the total number of predictions generated in each frame. More details on P will be provided in Section 5.1.2. B represents the training batch and λ_Ψ is the weighting factor for this loss term. λ_Ψ should be set such that Ψ is

roughly the same order of magnitude as the regression and classification losses, but slightly smaller. The idea is that we should not sacrifice regression and classification performance for improvements in XC.

Using the same set of notations, the following loss term aims to reduce XC values:

$$\tilde{\psi}_i = \sum_j (XC_j),\ \forall j \in P \tag{5.4}$$

$$\tilde{\Psi} = \lambda_{\tilde{\Psi}} * \frac{1}{|B|} \sum_i \tilde{\Psi}_i,\ \forall i \in B \tag{5.5}$$

The next loss term Ω aims to reduce PAP and make the attribution map smoother:

$$PAP_j = \sum_{m,n} \left|\phi^j_{m+1,n} - \phi^j_{m,n}\right| + \left|\phi^j_{m,n+1} - \phi^j_{m,n}\right| \tag{5.6}$$

$$\omega_i = \sum_j PAP_j,\ \forall j \in P \tag{5.7}$$

$$\Omega = \lambda_\Omega * \frac{1}{|B|} \sum_i \omega_i,\ \forall i \in B \tag{5.8}$$

PAP_j is the PAP value computed from the aggregated attribution map (single-channel attribution map generated by summing all positive/negative attributions at each pixel, see Section 3.3 for details) for the j^{th} prediction in a frame. m and n are pixel coordinates in the attribution map ϕ^j. ω_i is the PAP loss for the i^{th} LiDAR frame and Ω is the PAP-based loss for a batch. Again, λ_Ω is tuned such that Ω is of the same order of magnitude as the classification and regression losses.

5.1.2 Training strategies

We set $|P|$ to be three to avoid slowing down the training process too much. But we are reluctant to reduce $|P|$ any further since we do not wish to let only one or two predictions in each frame to dictate the attribution-based losses, which may impede the model's ability to generalize. Since this is the first attempt of using XC and PAP to train object detection model and we are not sure what will work, we propose different training strategies and list them out in this section. We try different ways of selecting predictions for P, as well as try increase/decrease the XC values for predictions in P.

XC_top_3_increase

One way is to select the three predictions with the highest top class scores in each frame as P. The reasoning is as follows:

1. The TP boxes tend to have higher XC values and it makes sense, because they match the type of ground truth objects which the model has been trained to detect, features learned from such objects should heavily influence model output (which leads to high attribution values within the bounding box and high XC). Perhaps making the XC values higher for the TP predictions can further boost model performance.

2. As shown in Figure 4.3, the predictions with relatively high top class score are most likely TP predictions.

Hence, by increasing the XC score for the predictions with high class score using Ψ (Equation (5.9)), we are increasing the XC score for a lot of the TP predictions. In other words, we are using a readily available value, the top class score, as an oracle that tells if a prediction has high/low XC score and if it is TP/FP.

XC_bottom_3_reduce

Another way to create P is to select the three predictions with the lowest top class scores. The reasoning is as follows:

1. The FP boxes tend to have lower XC values and it makes sense, because these objects do not correctly match any ground truth objects that the model has been trained to detect, the model should not be heavily influenced by features learned from these objects (low activation leads to low attribution values within the bounding box and low XC). Perhaps further lowering the XC values for the FP predictions can boost model performance.

2. As shown in Figure 4.3, the predictions with relatively low top class score are most likely FP predictions.

Hence, by reducing the XC score for the predictions with high class score using $\tilde{\Psi}$ (Equation (5.5)), we are reducing the XC score for a lot of the FP predictions. Perhaps making the model focus less on these irrelevant objects, the overall model performance can improve.

XC_bottom_3_increase

We also try increase the XC score for the predictions with low top class score using Ψ. The reasoning is that predictions with low top class score often have low XC too. If we

can increase XC for those predictions that tend to have low XC, then XC for the other predictions should rise too. Perhaps increase XC in general would be beneficial for model performance. Again, we are using the top class score as an oracle for XC values. This oracle will not be 100% correct but we need to rely on it due to how prohibitively costly it is to compute XC for all predictions.

XC_random_TP_FP

Instead of relying on the top class score as an oracle, we can also determine if a prediction is TP or FP during training time and apply different loss functions to them. We can try apply Ψ to three randomly selected TP predictions, and also apply $\tilde{\Psi}$ to three randomly selected FP predictions in each frame. The goal is to increase XC for the TP predictions and also reduce XC for the FP predictions. For the KITTI dataset, sometimes we may have less than three TP predictions in a frame. In such cases, we do not want the loss term to drop due to the lack of TP predictions. The loss term Ψ should drop only when XC improves, not when there are less TP predictions. Hence we normalize Ψ as if we have three TP predictions in each frame of the batch:

$$\Psi_{normalized} = \Psi * (|B| * |P|)/T \tag{5.9}$$

where T is the number of TP boxes in the batch for which XC-related loss is applied. For example, say the batch size is two, three TP boxes are chosen from the first frame, but the second frame contains only one TP box. Ψ is computed from only four TP boxes in the batch. Then we scale Ψ by a factor of $6/4$, because there would have been $2 \times 3 = 6$ TP boxes in this batch in the usual case.

PAP_bottom_3

We apply Ω (Equation (5.8)) to reduce PAP value for the attribution maps generated for the three predictions in each frame with the lowest top class score. Based on our observations, the FP predictions tend to have noisier attribution maps (which would lead to higher PAP values). Hence, if we can reduce PAP for those predictions which tend to have high PAP values (the FP predictions with low top class score), then PAP values for the other predictions should be reduced as well, and the model should become more robust.

PAP_top_3

We apply Ω to reduce PAP value for the attribution maps generated for the three predictions in each frame with the highest top class score. This training scheme serves as a comparison for PAP_bottom_3.

5.2 Implementation details

It is important to explain some details regarding our experiments before diving into the results. We adopt OpenPCDet's [54] default settings for training the models: the Adam optimizer is used and a schedule for the learning rate and the beta values is applied to the optimizer. As shown in Figure 5.1, the learning rate starts with a small value of 3×10^{-4} and gradually reaches a peak value of 3×10^{-3} around the middle of the training cycle, then eventually approaches 0 towards the end of the training cycle. The probable reasoning behind such design is such: initially, the loss would be high, hence the learning rate does not need to be large; but as the model performs better, the learning rate should increase to avoid getting stuck at some local minimum of the loss function; closer to the end of the cycle, only small fine tuning is desired, small learning rate is ideal for avoiding overfitting.

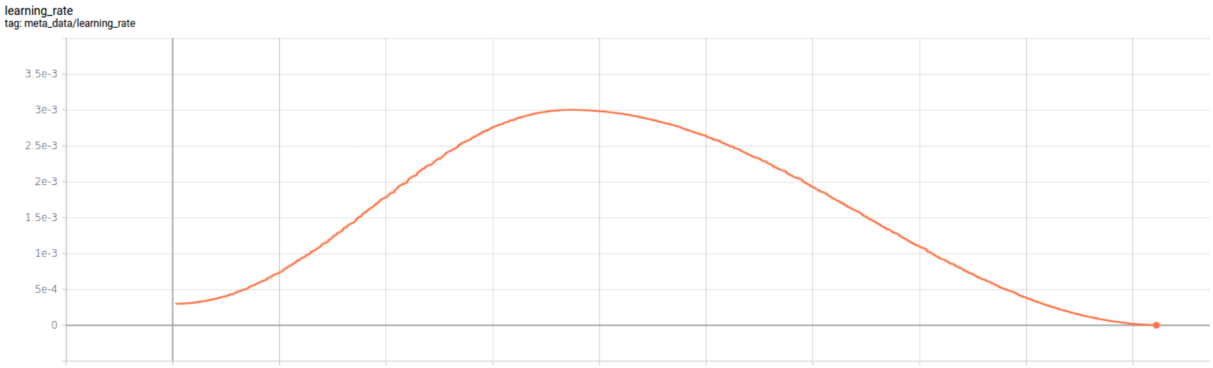


Figure 5.1: Default learning rate schedule for Adam optimizer in OpenPCDet.

Unless otherwise specified, we apply $\lambda_\Psi = \lambda_{\tilde{\Psi}} = 0.2$ for the XC-based loss terms, and $\lambda_\Omega = 5 \times 10^{-4}$ for the PAP-based loss terms. We use Nvidia Tesla v100-sxm2-32gb GPUs to train our models with these attribution-based losses with a batch size of 4. For all experiments, we train PointPillars on the KITTI dataset for 80 epochs. To rule out any effect on model performance due to the shuffling of LiDAR frames, we fix the batches: instead of shuffling the LiDAR frames differently each epoch, we only shuffle once and the same batch would always contain the same LiDAR frames.

Another detail worth mentioning is that we set a cap of 0.3 for all the attribution-based loss functions. The sum of the regression and classification losses is in the range of [0.4, 0.7] throughout most of the training cycle. Occasionally the attribution-based losses can spike high. To prevent them from overwhelming the other losses, we scale them down when they exceed 0.3.

5.3 Results on XC-based loss terms

We present our results on training with XC-based loss terms in Table 5.1. Due to the time consuming nature of these experiments ($\sim$ 2.5hrs per epoch), we have only explored loss terms based on XC_s^+. Training with XC generated by IG is even more costly ($\sim$ 7hrs per epoch). Thus, we have only done two experiments with IG and have to stop them early. The average precision (AP) values presented in Table 5.1 are averaged over corresponding precision values at 40 recall values equally spaced between 0 and 1. The presented AP values are in the range of [0, 100] instead of [0, 1] because they are in percentages. See Section 2.3.2 for more information on the AP metric for object detection.

Table 5.1: Training with XC-based loss terms on KITTI. The AP values for different object classes and difficulty levels are presented for model checkpoint saved at the 20^{th}, 40^{th}, 60^{th}, and 80^{th} epochs. Underline means less than one percent better than the baseline value, **bold** means one percent or more than the baseline value.

Epochs	Training Strategy	Easy			Medium			Hard		
		Car	Pedestrian	Cyclist	Car	Pedestrian	Cyclist	Car	Pedestrian	Cyclist
20	Regular	78.7	44.7	68.0	67.8	39.0	48.1	63.8	35.1	45.1
	XC_s^+_top_3_increase	69.9	40.6	56.4	59.5	36.5	41.0	57.2	33.2	38.4
	XC_s^+_bottom_3_increase	74.9	**47.7**	64.3	63.3	**41.6**	45.2	58.7	**37.6**	42.4
	XC_s^+_bottom_3_reduce	**83.6**	45.4	68.5	68.5	**40.1**	**50.7**	**65.4**	**36.3**	**47.7**
	XC_s^+_random_TP_FP	74.0	39.9	66.2	60.9	34.7	46.2	58.6	31.0	43.2
	XC_s^+_top_3_increase$_{IG}$	62.4	29.0	60.7	56.7	26.0	39.7	55.3	23.5	37.1
	XC_s^+_bottom_3_increase$_{IG}$	64.4	28.6	52.6	52.1	26.8	34.3	51.0	24.5	32.2
40	Regular	84.4	55.6	77.5	73.3	49.6	57.6	69.1	45.6	54.4
	XC_s^+_top_3_increase	66.8	45.9	52.3	56.5	41.1	38.2	53.5	36.8	36.1
	XC_s^+_bottom_3_increase	**86.0**	49.3	71.6	72.7	42.8	51.1	69.0	38.5	47.7
	XC_s^+_bottom_3_reduce	83.4	52.4	74.4	71.1	46.1	56.0	66.3	41.9	52.9
	XC_s^+_random_TP_FP	76.7	52.0	75.4	67.2	46.7	56.9	64.4	42.3	53.3
	XC_s^+_top_3_increase$_{IG}$	73.4	41.4	60.3	63.1	36.2	39.3	59.4	32.4	37.0
	XC_s^+_bottom_3_increase$_{IG}$	59.6	38.0	67.2	50.1	34.6	42.9	46.8	30.8	40.0
60	Regular	86.1	51.2	73.0	74.2	46.3	57.1	71.6	41.9	53.4
	XC_s^+_top_3_increase	80.8	44.9	73.5	71.5	39.7	55.1	69.1	35.9	51.3
	XC_s^+_bottom_3_increase	86.2	47.9	**76.2**	74.9	41.9	55.0	71.9	38.0	51.2
	XC_s^+_bottom_3_reduce	86.1	51.6	**77.9**	74.7	46.0	**59.3**	71.5	41.6	**55.1**
	XC_s^+_random_TP_FP	80.8	49.9	**76.3**	70.6	43.3	57.1	68.2	39.3	53.4
80	Regular	87.4	55.5	81.5	78.2	49.7	64.5	75.3	45.5	60.7
	XC_s^+_top_3_increase	83.1	41.3	74.1	74.6	37.0	56.6	72.3	33.9	53.0
	XC_s^+_bottom_3_increase	86.7	48.4	74.2	77.7	42.6	55.9	75.0	38.7	52.2
	XC_s^+_bottom_3_reduce	87.6	53.1	**84.2**	78.3	47.5	65.0	75.2	43.5	60.6
	XC_s^+_random_TP_FP	85.6	53.7	78.3	76.0	46.8	60.5	73.8	42.5	56.5

We also present result from training without any attribution-based losses with default settings from OpenPCDet (labelled as "Regular" in Table 5.1), with only one change: the frames in each batch are fixed to match the setting for all other experiments. This will serve as a baseline for comparison.

For all checkpoints evaluated, XC_s^+_top_3_increase is unable to beat the baseline on any object classes, except for a very small improvement on easy cyclist objects at the 60^{th} epoch. The XC values for the predictions with high top class scores are already high, perhaps trying to further increase them would not lead to any improvement. Rather, it would impede the effects of the classification and regularization losses and degrade model performance.

On the other hand, XC_s^+_bottom_3_increase is certainly doing a bit better, showing that increasing XC for the predictions with low top class score (also likely to have low XC) would be better than increasing XC for those with high top class score. However, this strategy is still unable to outperform the baseline for most object classes. It is possible that for most TP objects, since the model has been trained to detect them and is familiar with them, features within the object boundaries are enough for the model to make high confidence predictions, leading to high class score and high XC; whereas for the FP objects and grouth objects that look like FP objects, the model is not certain about them and needs to rely more on the context to make predictions, leading to lower class score and lower XC. Making the XC higher for these uncertain objects would force the model to pay less attention to the context, depriving the model of some useful information that can help generating correct predictions. In short, increasing XC for the low confidence predictions can hurt model performance, but increasing XC for the high confidence predictions is even more detrimental.

XC_s^+_bottom_3_reduce is able to produce the most promising results out of all seven experiments on XC-based losses. At the 20^{th} epoch, it outperforms baseline on all nine object classes, six of which are more than 1% improvement. At the 60^{th} epoch, it performs slightly better than the baseline and at the 80^{th} epoch it performs slightly worse. These observations support the hypotheses in the last paragraph: the model needs context information for uncertain objects, reducing XC for such objects can encourage the model to use more context information and improve performance. A worth-noting observation is that at the 40^{th} epoch, XC_s^+_bottom_3_reduce performs notably worse than the baseline. It might be due to OpenPCDet's learning rate schedule (Figure 5.1). The 40^{th} epoch is in the middle of the training cycle, where the learning rate is relatively high. But in early epochs, the learning rate is small. It is possible that the success of XC_s^+_bottom_3_reduce at the 20^{th} and 60^{th} epochs is partly due to the small learning rate, and at the 40^{th} epoch. It is possible that the high learning rate makes the XC loss influence the model too much and

degrades performance.

XC_s^+_random_TP_FP is performing better than XC_s^+_top_3_increase but worse than XC_s^+_bottom_3_reduce on most object classes. Note that the batch size for this experiment is set to three instead of four due to the increased memory demand resulting from the additional number of predictions for which we are computing XC. Overall, it is performing worse than the baseline. This is understandable as we have already seen that increasing XC for the high confidence predictions is detrimental, but increasing XC for the low confidence predictions is beneficial. If we combine these two schemes, the result is expected to be somewhere in the middle.

An interesting observation is that, just as the IG-based XC scores can not distinguish the TP and FP predictions than the backprop-based XC scores do (see Section 4.4), they most likely can't lead to better AP either when incorporated in the training process. At least when training with XC_s^+, using IG does not lead to any notable improvements compared to using backprop. It even leads to very notable degradation for the car and pedestrian objects.

5.4 Results on PAP-based loss terms

The results of the experiment on PAP-based loss terms are presented in Table 5.2. The "+" and "-" superscripts indicate whether the positive or negative aggregated attribution map is being used to compute PAP. It is clear that the PAP-based loss terms are doing better than the XC-based loss terms. The only XC-based training strategy which leads to notable improvements compared to the baseline is XC_s^+_bottom_3_reduce, but all experiments using PAP-based loss lead to improvements at the 20^{th} and 60^{th} epochs. Erion *et al.* [11] showed that although PAP-based loss can improve performance on MNIST dataset with additional Gaussian noise, it led to worse performance on the original MNIST dataset compared to training without PAP. This is perhaps because the MNIST digit classification task is a very simple task, the model does not need to have high generalization capabilities anyways, increasing model robustness would not improve model performance unless the data contains extra noise. But LiDAR object detection is a much more difficult task, and PointPillars must contain a lot more parameters than a simple digit classifier. For PointPillars, the ability to generalize is highly desired, we can therefore improve model performance using PAP even on the uncorrupted original KITTI dataset.

When training with PAP, the preferred selection of predictions is different than when training with XC. It is very clear that applying XC-based loss to the low confidence predictions is more beneficial than applying them to the high confidence predictions. For

each experiment, we present 36 AP metrics (nine object classes and four checkpoints). XC_s^+_bottom_3_increase beats XC_s^+_top_3_increase in 32 out of the 36 metrics, and XC_s^+_bottom_3_reduce beats XC_s^+_top_3_increase in all 36 metrics. However, when training with PAP-based loss, it seems that applying it to the high confidence predictions might be more beneficial. PAP^+_top_3 beats PAP^+_bottom_3 in 27 AP metrics and PAP^-_top_3 beats PAP^-_bottom_3 in 24 AP metrics. But we do need to acknowledge that the difference in AP due to the selection of predictions is less for PAP than for XC. Further experimentation is required for a more concrete conclusion.

Table 5.2: Training with PAP-based loss terms on KITTI. Underline means less than one percent better than baseline value, **bold** means one percent or more than basline value.

Epochs	Training Strategy	Easy			Medium			Hard		
		Car	Pedestrian	Cyclist	Car	Pedestrian	Cyclist	Car	Pedestrian	Cyclist
20	Regular	78.7	44.7	68.0	67.8	39.0	48.1	63.8	35.1	45.1
	PAP^+_top_3	**82.4**	44.9	**74.5**	**70.2**	**40.5**	**50.8**	**67.5**	**37.0**	**47.7**
	PAP^+_bottom_3	**82.3**	**47.9**	**74.5**	**68.2**	**41.8**	**52.3**	64.3	**37.6**	**48.8**
	PAP^-_top_3	**80.3**	**47.7**	**77.3**	67.5	**42.0**	**55.4**	62.7	**37.9**	**51.7**
	PAP^-_bottom_3	**82.0**	**49.5**	**74.1**	68.6	**43.2**	**51.0**	**65.2**	**38.8**	**48.0**
40	Regular	84.4	55.6	77.5	73.3	49.6	57.6	69.1	45.6	54.4
	PAP^+_top_3	69.0	51.5	78.3	58.4	45.9	54.4	54.9	41.6	51.0
	PAP^+_bottom_3	72.3	50.2	73.0	57.0	44.3	52.1	53.5	39.3	48.5
	PAP^-_top_3	82.1	49.5	**80.1**	66.1	43.9	**58.6**	62.7	39.5	54.6
	PAP^-_bottom_3	78.0	52.3	70.3	62.4	46.2	51.4	58.8	41.3	48.0
60	Regular	86.1	51.2	73.0	74.2	46.3	57.1	71.6	41.9	53.4
	PAP^+_top_3	86.5	**52.9**	**83.3**	74.9	47.3	**63.3**	71.7	42.8	**59.2**
	PAP^+_bottom_3	86.6	52.1	**76.2**	74.8	47.2	57.7	71.8	42.7	53.9
	PAP^-_top_3	84.6	**54.0**	**80.9**	73.6	**48.2**	**60.4**	70.8	**43.1**	**56.4**
	PAP^-_bottom_3	86.1	52.1	**76.0**	75.0	46.5	56.3	72.2	42.2	52.6
80	Regular	87.4	55.5	81.5	78.2	49.7	64.5	75.3	45.5	60.7
	PAP^+_top_3	87.6	54.9	**83.7**	78.2	49.1	65.4	75.2	44.8	61.2
	PAP^+_bottom_3	87.2	53.6	78.9	77.6	47.9	61.0	75.0	43.7	56.9
	PAP^-_top_3	87.6	54.6	**85.1**	78.5	49.2	**65.8**	75.4	44.7	61.3
	PAP^-_bottom_3	87.5	54.3	81.3	77.9	48.1	62.5	75.2	44.1	58.4

Compare to the baseline, PAP^-_top_3 performs particularly well on the cyclist objects. Maybe the attribution-based loss is applied to more cyclist predictions in PAP^-_top_3 than in other experiments. To see if this is really the case, we present the number of predictions in each object class for which the attribution-based losses have been applied for three experiments at four different checkpoints in Table 5.3. Although PAP^-_top_3 leads to

notably better performance on the cyclist objects than the other two experiments, the number of cyclist predictions for which we apply the attribution-based losses is similar to the other two experiments. Also, despite that a lot less pedestrian predictions are used for PAP^{-}_top_3, its performance on the pedestrian object isn't much worse than the other two strategies. Hence, it is highly unlikely that the improvements on some specific object classes are due to large number of objects from that class being coincidentally selected for attribution-based loss computation.

Table 5.3: Number of predicted objects for which the attribution-based losses have been applied. Each count value is aggregated over a twenty-epoch interval.

Epochs	Training Strategy	Car	Pedestrian	Cyclist
1 to 20	PAP^{-}_top_3	163247	19606	39867
	PAP^{-}_bottom_3	87964	92780	41976
	XC_s^{+}_bottom_3_reduce	101746	75640	45334
21 to 40	PAP^{-}_top_3	164206	20179	38335
	PAP^{-}_bottom_3	82643	100304	39773
	XC_s^{+}_bottom_3_reduce	98785	86815	37120
41 to 60	PAP^{-}_top_3	171256	16142	35322
	PAP^{-}_bottom_3	78299	109234	35187
	XC_s^{+}_bottom_3_reduce	93322	95661	33737
61 to 80	PAP^{-}_top_3	175689	12518	34513
	PAP^{-}_bottom_3	73109	116730	32881
	XC_s^{+}_bottom_3_reduce	86523	104311	31886

Another important observation is that the PAP-based losses can led to better results than the baseline at the 20^{th} and the 60^{th} epochs, but similar results to the baseline at the end of training cycle and usually worse results than the baseline in the middle of the training cycle where the learning rate peaks. We have observed similar trend for XC_s^{+}_bottom_3_reduce. We now have more confidence in our hypothesis that only small weighting factors should be applied to the attribution-based losses. Close to the middle of the training cycle, the learning rate is higher. As a result, the attribution-based losses would have heavier influence on training, which may not be beneficial.

Training with PAP is promising, especially PAP^{-}_top_3: it beats the baseline on six out of nine object classes at the 80^{th} where most other training strategy perform either

similar to the baseline or slightly worse, and it even beats the baseline on three object classes at the 40^{th} epoch, where most other training strategies are doing worse than the baseline.

Chapter 6

Conclusion and future works

To use the explanations quantitatively, we proposed four XC scores for object detectors to measure the concentration of the feature attribution values generated for individual predictions. The XC scores can be computed for any object detector that has a voxelized representation of the 3D point cloud, and with any explanation method that produces feature attributions as explanations. Applying the four XC scores on the task of classifying TP vs. FP predictions led to over 100% improvement in AUPR on the pedestrian class on both the KITTI and the Waymo datasets compared to simple heuristics such as distance to sensor and number of LiDAR points in bounding box. Although the XC scores alone could not outperform class score in the TP vs. FP classification task, combining class score with the XC scores using an MLP led to notable improvement compared to using class score alone on the pedestrian predictions.

We also discovered that the XC metrics derived from backprop attributions often outperform those derived from IG attributions on TP vs. FP classification for the pedestrian and cyclist objects. This indicates that explanations that are more understandable to a human observer, such as IG, may not offer superior value when analyzed quantitatively and researchers should not discard simpler explanation methods when exploring quantitative use cases for XAI.

We experimented with loss terms based on XC and PAP computed from backprop attributions. We discovered that further decreasing XC values for the low confidence predictions can lead to improvement in model performance. But all other training strategies based on XC failed to produce any notable improvement. On the other hand, decreasing PAP for the high confidence predictions seemed to be a promising strategy for boosting model performance, especially for the cyclist objects. For all these experiment, there was

a reoccurring observation: the benefit offered by the attribution-based loss terms are the most notable when learning rate is around 0.0015, but as learning rate gets higher, these loss terms often degrade model performance. For 3D object detection, we believe it is possible to achieve model performance boost large enough to justify the extra computational cost of training with attributions. Perhaps lowering the weighting factor for attribution-based losses when learning rate is higher can be beneficial. Or there could be a better way for selecting predictions for which we compute the attribution-related losses, rather than just greedily choosing the most/least confident predictions.

There are still few limitations with this work which could be addressed by future work. Firstly, although we had experimented with multiple datasets (KITTI and Waymo), we had only experimented with one model, PointPillars. In the future, it would be interesting to apply the XC scores to predictions generated by other object detectors as well and see if the XC scores still offer the same benefits. Secondly, we had only explored two XAI methods, IG and backprop. Backprop is rather crude, thus IG is the only sophisticated gradient-based explanation we had analyzed. It would be interesting to compare XC scores produced by other XAI methods such as guided-backprop or Grad-CAM as well. Thirdly, we had only analyzed car, pedestrian, and cyclist predictions. Future researchers are encouraged to further explore XC scores or PAP for other types of objects as well.

References

[1] Alejandro Barredo Arrieta, Natalia Diaz-Rodriguez, Javier Del Sera, Adrien Bennetotb, Siham Tabik, Alberto Barbadoh, Salvador Garciag, Sergio Gil-Lopeza, Daniel Molinag, Richard Benjaminsh, Raja Chatilaf, and Francisco Herrerag. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. In *Information Fusion*, 2019.

[2] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[3] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[4] Weijie Chen, Yilu Guo, Shicai Yang, Zhaoyang Li, Zhenxin Ma, Binbin Chen, Long Zhao, Di Xie, Shiliang Pu, and Yueting Zhuang. Box re-ranking: Unsupervised false positive suppression for domain adaptive pedestrian detection. *arXiv preprint arXiv:2102.00595*, 2021.

[5] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, July 2017.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

[7] Krzysztof Czarnecki. Lecture notes in ECE495, February 2021.

[8] Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. In *ICML*, 2006.

[9] Yadolah Dodge. *The Concise Encyclopedia of Statistics*. Springer, 2008.

[10] Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. Learning credible deep neural networks with rationale regularization. *arXiv preprint arXiv:1908.05601*, 2019.

[11] Gabriel Erion, Joseph D. Janizek, Pascal Sturmfels, Scott Lundberg, and Su-In Lee. Improving performance of deep learning models with axiomatic attribution priors and expected gradients. *arXiv preprint arXiv:1906.10670*, 2020.

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results. http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

[13] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

[14] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *CVPR*, 2012.

[15] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[16] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *ICCV*, Oct 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[19] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*, Feb 2017.

[20] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. In *ICCV*, 2017.

[21] Jinkyu Kim, Teruhisa Misu, Yi-Ting Chen, Ashish Tawari, and John Canny. Grounding human-to-vehicle advice for self-driving vehicles. In *CVPR*, pages 10591–10599, 2019.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2017.

[23] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2009.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[25] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, June 2019.

[26] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[27] Yann LeCun, Patrick Haffner, Leon Bottou, and Yoshua Bengio. Object recognition with gradient based learning. 1999.

[28] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *NeurIPS*, Dec 2018.

[29] Xiao-Hui Li, Caleb Chen Cao, Yuhan Shi, Wei Bai, Han Gao, Luyu Qiu, Cong Wang, Yuanyuan Gao, Shenjia Zhang, Xue Xue, and Lei Chen. A survey of data-driven and knowledge-aware explainable AI. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[30] Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *ICLR*, April 2018.

[31] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *arXiv preprint arXiv:1405.0312*.

[32] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *ECCV*, volume 9905, pages 21–37. Springer, 2016.

[33] Julia Lust and Alexandru P. Condurache. Gran: An efficient gradient-norm based detector for adversarial and misclassified examples. In *ESANN*, Oct 2020.

[34] Xingjun Ma, Bo Li, Yisen Wang, Sarah Erfani, Sudanthi Wijewickrema, Michael Houle, Grant Schoenebeck, Dawn Song, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. In *ICLR*, April 2018.

[35] Chris Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *NeurIPS*, pages 8024–8035. 2019.

[37] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, July 2017.

[38] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, Dec 2017.

[39] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[40] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

[41] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135—1144, 2016.

[42] Laura Rieger, Chandan Singh, W. James Murdoch, and Bin Yu. Interpretations are useful: penalizing explanations to align neural networks with prior knowledge. *arXiv preprint arXiv:1909.13584*, 2020.

[43] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2662–2670, 2017.

[44] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *ICCV*, Oct 2017.

[45] Rakshith Shetty, Bernt Schiele, and Mario Fritz. Not using the car to see the sidewalk — quantifying and controlling the effects of context in classification and segmentation. *arXiv preprint arXiv:1812.06707*, 2018.

[46] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *CVPR*, June 2020.

[47] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3d object proposal generation and detection from point cloud. In *CVPR*, June 2019.

[48] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2014.

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[50] Facebook Open Source. Captum: Model interpretability for pytorch. https://github.com/pytorch/captum, 2020.

[51] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2015.

[52] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Sheng Zhao, Shuyang Cheng, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. *CoRR*, abs/1912.04838, 2020.

[53] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *ICML*, 2017.

[54] OpenPCDet Development Team. OpenPCDet: An open-source toolbox for 3d object detection from point clouds. https://github.com/open-mmlab/OpenPCDet, 2020.

[55] E. Walter. Cambridge advanced learner's dictionary, 2008.

[56] Yuwen Xiong, Mengye Ren, Renjie Liao, Kelvin Wong, and Raquel Urtasun. Deformable filter convolution for point cloud reasoning, 2019.

[57] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *NDSS*, Feb 2018.

[58] Yiran Xu, Xiaoyin Yang, Lihang Gong, Hsuan-Chu Lin, Tz-Ying Wu, Yunsheng Li, and Nuno Vasconcelos. Explainable object-induced action decision for autonomous vehicles. In *CVPR*, pages 9523–9532, 2020.

[59] Yan Yan, Yuxin Mao, and Bo Li. Second: Sparsely embedded convolutional detection. In *Sensors*, 2018.

[60] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *CVPR*, June 2018.

[61] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *arXiv preprint arXiv:1311.2901*, 2013.

[62] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, June 2018.