Object Detection in Nighttime

(Public Version)

Author: Artificial-Rookie

Date: April 26th 2023

Please feel free to use the code or the report if you need it!

Abstract

Object detection is a very challenging and prevailing task for computer vision area. Meanwhile, although deep learning models have achieved state-of-the-art results on well-developed datasets, the limited availability of data and imbalanced classes can severely bias model predictions in real-world scenarios. In recent years, various works has been proposed and plenty of them achieved state-of-art results using fully-trained deep learning model and well-developed dataset. However, in real cases, the available data for model training could possibly be insufficient or imbalanced (maybe few-shot cases), which could cause sever biases when the model do the predictions during testing.

To address this problem, this work tried to implement a method based on meta-learning to alleviate the influence of imbalanced data to the tendency of the model's prediction. The experiments are implemented on BDD100K dataset which focus on transportation vehicles with city/countryside backgrounds. Daytime images with labeled bounding boxes are used to train the model and nighttime images with no labeling are used for testing. The work uses a two-stage training approach using two further developed datasets generated from the original one and achieved remarkable improvement with regard to the original one. These results suggest that this could be a worth-researching direction for object detection in nighttime when solving the imbalanced data problem.

List of Tables

List of Figures

Table of Content

Chapter 1 Introduction

1.1 Problem and Motivation

Object detection is a task involving simultaneous estimation of categories and locations of object instances in a given image. Networks and models used for object detection usually need large datasets for training to get good results. The data-driven approach requires the researchers to guarantee the dataset to be not only large, but also balanced and clean. However, making dataset large and balanced at the same time could be an exceedingly hard work. Data for different objects could be hard to collect, which leads to the imbalanced distribution of object features in the feature space.

The imbalance problem could generally be classified into four main types [1]: class imbalance, scale imbalance, spatial imbalance, objective imbalance. Class imbalance could occur when there is significant inequality among the sizes of different object categories. Then, scale imbalance focuses on object scales. Spatial imbalance focuses on mainly bounding box regressions (i.e., regression penalty, location, IoU etc.). Objective imbalance focuses on the balance of different loss functions between classification and localization (i.e., whether one of them dominates). Relations between different difficulties and the input properties they concern are listed in table 1.1 [1].

The problem we are trying to deal with during this project generally belongs to the field of class imbalance. Moreover, class imbalance could also be further divided into two aspects, which are foreground-background imbalance problem and foreground-foreground imbalance problem. The latter one is the task we are facing at during this project. This is the problem that the foreground classes include both over-represented and the under-represented classes. Once the data samples go through the feature extractor, the model it learns could possibly be biased. Thus, this has become a widely concerned aspect in object detection that is still remained to be not thoroughly studied and investigated. The following paragraphs will briefly introduce how the problem of the dataset affects the work of object detection, as well as the approach we use to address the task.

The distribution of data samples in the nature generally follows a power law, which means the object frequency in the real world is actually long-tailed [2] with regard to different categories. In our particular case, the classes that dominate the distribution in BDD100k (shown in Figure 1.1) could be cars, pedestrians, traffic signs, traffic lights etc. Even though in most cases, the majority classes could contain abundant and diverse training examples that well represent the expected data during inference time, the minority classes usually lack representative training data and thus makes the modeling biased and inaccurate.

The problem we are facing in this project is that we have imbalanced daytime training dataset and need to train the model to adaptively recognize same classes of objects in nighttime. There are no nighttime images in the training set, therefore the

problem requires the combination of object detection and domain adaptation. Furthermore, since the imbalance problem mainly affects the accuracy of image classification, here we choose to focus more on the classification part within the object detection rather than the localization part.

Type	Imbalance Problem	Related Input Properties
Class	Foreground-Background	The numbers of input bounding boxes pertaining to different classes
	Foreground-Foreground	
Scale	Object/Box Level	The scales of input and ground-truth bounding boxes
	Feature Level	Contribution of the feature layer from different abstraction levels of the backbone network (i.e., high and low level)
Spatial	Regression Loss	Contribution of the individual examples to the regression loss
	IoU Distribution	IoU distribution of positive input bounding boxes
	Object Location	Locations of the objects throughout the image
Objective	Classification & Localization	Contribution of different tasks (i.e., classification, regression) to the overall loss

Table 1.1 Categories of Imbalance Problems and Related Input Properties

1.2 Assumptions and Backgrounds

In this project, we focus on the classification part among the whole picture of object detection in nighttime. The goal of the project is to use deep learning as well as data processing methods to make the model to have less tendencies to classify an object as one of the majority classes. Traditional methods which do not involve any balancing technique use simply cross-entropy losses to train neural networks. However, in order to develop a method that could be used in most of the common and natural cases, the testing datasets do not usually follow the identical data distribution as the training set. The most obvious solution is to adapt the model from the source domain to the target domain. The testing data we used in this project is actually balanced with regard to each category, through which the imbalanced problem caused by the training set could be revealed thoroughly.

The method involved within this project used meta-learning to fine-tune the model

trained by ResNet32. The training process is divided into two stages, the first is just vanilla training with ResNet32, while the second stage is using a meta-model which is also ResNet32 with a different curated dataset for balanced training. The first stage could be considered as pre-training and to obtain a biased model. Then the second stage is designed to model the biases and alleviate the impact it caused to the model.

Denote the distributions of objects in source and target domain [2] as $P_s(x, y)$ and $P_t(x, y)$. For most of the methods, we have the assumption: $P_s(x, y) \neq P_t(x, y)$, which is also the motivation of dealing with class imbalance problem. Weighting methods with regard to such problems are generally allowing the minority classes representing bigger areas in the feature space and hoping the enlarged space could include more potential data samples. However, plenty of conventional methods are in fact designed under the prerequisite, i.e., the training data and the testing data have same data distributions w.r.t. the same object class. For example, for target-shifts, the difference is only the number of samples in each class which means $P_s(y) \neq P_t(y)$, but the equation $P_s(x|y) = P_t(x|y)$ holds. Here, we do update of such assumption, because believe that the equation $P_s(x|y) = P_t(x|y)$ does not hold [2] when the domain shifts from source to target. For instance, in city areas the distribution of pedestrians or cars should be different from that of countryside, daytime objects should also be different from nighttime ones.

As stated in the previous paragraph, the data in the source domain (training set) is not well represented due to the lack of samples in minority classes. Consequently, only the majority class distributions could be approximated using the samples in the source domain. The solution attempted in this project inherited the class-wise weights and then further explored the sample-wise weights by specifically modelling the conditional probability changes in the target domain. We developed a way to curate the dataset to implement this strategy in object detection framework and provided insights for current problems and future plans.

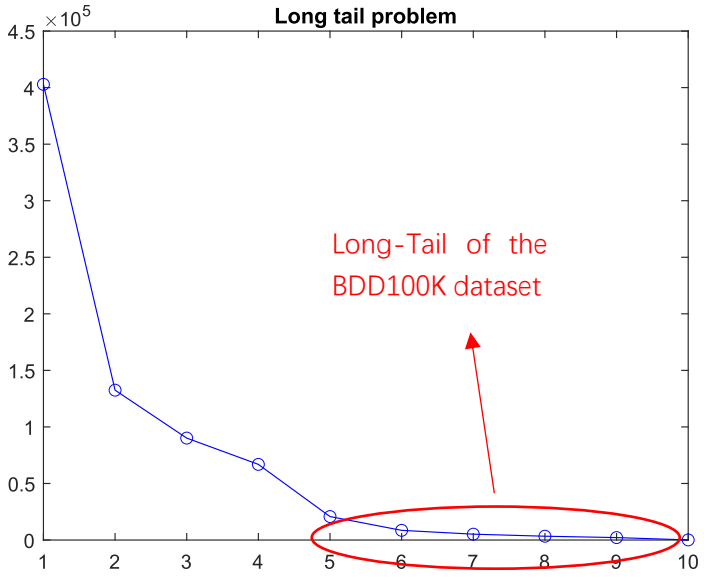

Figure 1.1 Long-tail Problem in BDD100K

Chapter 2 Related Works

2.1 Object Detection

In the history of computer vision, there is no doubt the rise of CNN has led to a swift paradigm shift from the traditional algorithms to deep neural networks. Then, a large number of works emerged continuously. Among which, R-CNN has gained a great success by proposing a method to generate region proposals and use a network to generate RoI (region of interest) independently [4]. Then, the appearance of Faster R-CNN introduced RPN (region proposal network) [5] which gives high flexibility and remarkable results. Basically, the main-stream method for object detection is proposing a region in which the system decides it contains background or objects. Then, regression methods are used to optimize the bounding boxes. The classification stage is the last stage, which performed within the decided bounding boxes.

Though these former works are considering neither domain adaptation tasks nor class imbalance tasks, there are still a lot of ideas that are worth learning from. The whole object detection system for nighttime could be built by combining the classification part in this project and the localization part from Faster R-CNN.

2.2 Imbalanced Data

The imbalanced data problem in object detection could be categorized into many types, the one we are dealing with in this project is the foreground-foreground imbalanced problem (i.e., samples in a few classes dominates the others). There are mainly three types of solutions, resampling methods, reweighting methods and generative methods. Several other approaches that we also tried to learn from are not belong to any of these three main types, but will also be listed in the “others” type to provide some insight of this problem.

2.2.1 Resampling

Basically, resampling methods could be further divided in to two approaches, over-sampling and under-sampling. Over-sampling focuses on minority classes by continuously adding repeated samples into the training dataset [3], so that the minority classes are manually enlarged, with the cost that the model would easily overfit. On the other hand, under-sampling is to remove random samples from the majority classes and simultaneously hoping that these does not affect the whole representation of data distribution to much. The comparison between over-sampling and under-sampling about which one outperforms the other still remains to be controversial.

2.2.2 Reweighting

The idea of reweighting methods could be very straightforward, but the assumptions and mathematical proves in behind could be very complicated. The idea is that for different classes, different weights are assigned during classification process (i.e., larger classes have smaller weights and vice versa). In this way, minority classes could possibly have better chance to be recognized. Nevertheless, the determination of the reweighting term is a great challenge. Naively, the weights could set as the reciprocal of the number of samples in the class. Based on this intuitive idea, plenty of approaches are proposed and most of them could outperform the naive method.

There was a rather intriguing and brilliant cost-sensitive reweighting approach proposed in [6], with which the effect is shown in figure 2.1. The author proposed a new loss used in the balancing training phase that alleviates the influence of the samples that cause an overfitted decision boundary. Another prevailing method is using the Focal loss proposed by Tsung-Yi Lin et.al. in 2018, is a dynamically scaled cross-entropy loss [8], where the scaling factor decays to zero as confidence in the correct class increases as shown in Figure 2.2 below. The focal loss is somehow similar to the effective number reweighting approach which we use in this project, the way it reweights the

2.2.3 Generative

Generative methods in fact partially overlaps with the resampling methods. This method is meant to generate artificial samples to balance the training. The most representative one is to use GAN or further developed works (generative adversarial network) to enlarge the dataset we have. Though, generative methods do not suit the case of domain adaptation, because we do not have access to nighttime images, so the best we could try is data augmentation (e.g., transfer the daytime image to somehow approximate nighttime as illustrated in [7]). It is also possible to try cropping some of the objects with fair resize operations and then fit them into other backgrounds with some smoothing techniques such as passing through gaussian filters as suggested in [7]. This might be a feasible way of “faking data” but there is still a risk of overfitting, because even with the background has changed, the feature of the object is basically the same.

2.2.4 Others

There are still several approaches that do not belong to any of the above categories. Using prototype to model the data is a rather interesting approach to learn object features, which is also a valuable method that we could learn from. In [9], the author proposed a method to model the data by universal prototype, which finds and enhances the features with intrinsic characteristics across different categories. According to the

author, this helps the model to capture class invariant characteristics and reduce the impact of data imbalance across different categories. The author also mentioned that meta-learning approaches (e.g., the method we used) could have severe generalization problems, which we must pay attention to during experiments.

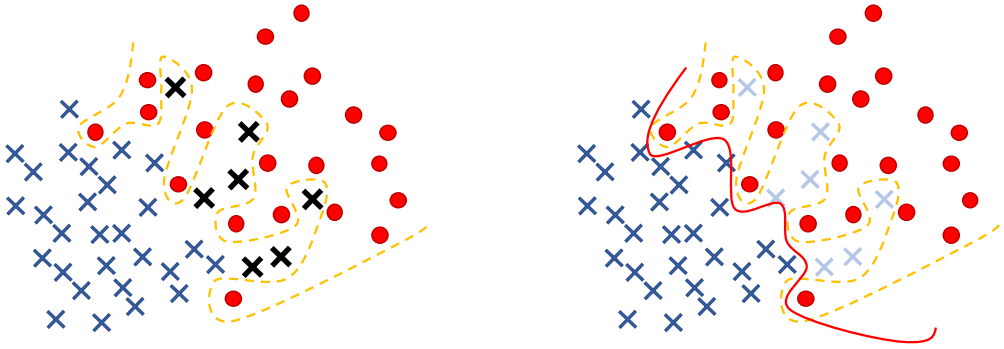

Figure 2.1 Cost-sensitive Reweighting Method [6]

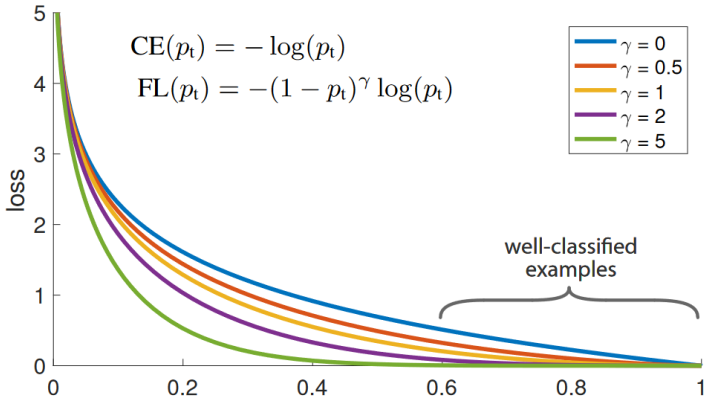

Figure 2.2 Probability of Ground Truth Classes [8]

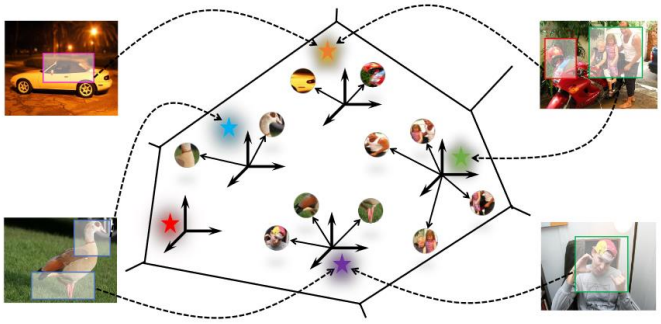

Figure 2.3 Universal Prototypes Learned from All Classes [9]

2.3 Domain Adaptation

As stated in the first chapter, the training data (source domain) and the testing data

(target domain) could follow different data distributions which makes the model perform far from expectations during testing. Collecting data from the target domain indeed helps to improve the performance, but also requires extra time and financial consumptions. To alleviate the performance drop cause by domain shifts, various solutions are proposed. In [10], they use the idea of adding physical knowledge into the object detection model, which helps to capture some domain invariant features. This is a very popular and effective way of solving such problems. Such prior knowledge teaches the model which features it should pay attention to, so that even after the domain shifts, such features could still be recognized and emphasized by the model.

In [11], disentangled learning is used to recognize different features of the same class. This is also an approach to learn from domain invariant features, but the point is that it focuses more on the instance level representations instead of holistic and abstract representations which is often the case for merely image classification. Such methods are quite possible to be used together with our work. However, due to limited time and resource, hopefully the ideas could be combined into the system in near future.

Chapter 3 Approaches and Methodologies

3.1 Modeling Class-wise Weights and Conditional Weights

In this chapter, more detailed mathematical derivations will be provided to show how the bias is generated, as well as the way network fine-tunes itself. Here, we continue to use the expressions in chapter 1, i.e., $P_s(x, y)$ and $P_t(x, y)$. The marginal distribution $P_s(y)$ indicates the distribution of sample numbers between classes in the source domain, which is also the biggest difference between head classes and tail classes (see in Figure 3.1 the data distribution for BDD100K).

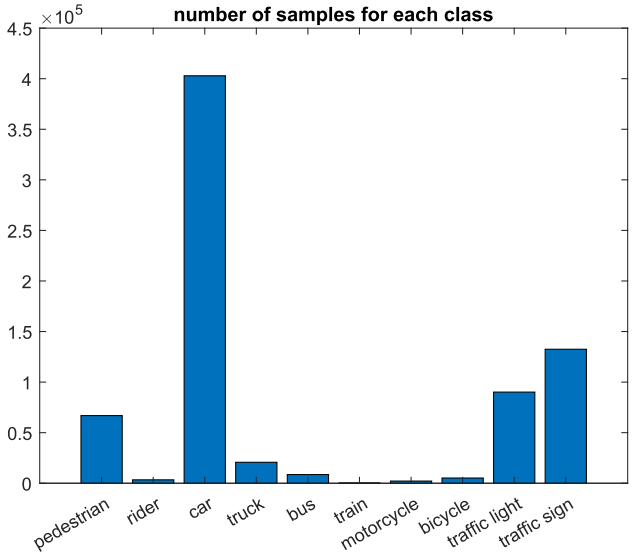

Figure 3.1 Number of Samples in Each Class in BDD100K

From [2] we borrow the proposed error definition:

$$error = E_{P_t(x,y)} L(f(x;\theta), y) \tag{3.1}$$

The L here stands for the 0-1 loss function, but in order to get the whole system differentiable, this could be replaced with the cross-entropy loss. Our expectation of the target domain is that it could provide a more balanced data distribution. In order to relate the source and the target domain, using the conversion rule of the expectation, we could rewrite equation 3.1 as:

$$error = E_{P_s(x,y)} L(f(x;\theta), y) P_t(x, y)/P_s(x, y) \tag{3.2}$$

Furthermore, the joint probabilities could be separated into conditional probability times marginal probability. Denote the weights for each class as w (class-wise weights) and the bias caused by domain transition as ε (conditional weights). By definition, there are:

$$w = P_t(y)/P_s(y) \qquad \varepsilon = w * (\frac{P_t(x|y)}{P_s(x|y)} - 1)$$

With the use of these annotations, equation 3.2 could be further rewrite as:

$$error = E_{P_s(x,y)} L(f(x;\theta), y)(w_y + \varepsilon_{x,y}) \tag{3.3}$$

The class-wise weight w is one that conventional weighting methods propose to calculate, which believes the conditional distribution should equal to 1 (i.e., $\varepsilon = 0$ in equation 3.3). Since in domain adaptation cases the equation does not hold, modeling the parameter ε becomes a very important task to address the imbalance problem. The approach used in this project modeled both the w and the ε, so that the weight we assign to each data sample would be the combination of these two parts. This requires us to estimate both parameters in practice, which would be illustrated in the following sections respectively.

3.2 Class-wise Weights

As stated in the previous paragraphs, the imbalanced data problem exists due to the wrong or lack of representation of data distribution. To address this problem, we need to go back to how the data samples fill the feature space of a particular class. Now suppose we are focusing on only one class with a volume of N. The ideal dataset should be a subset sampled from N that could densely and evenly fill the feature space of the class. From this perspective, each of the data sample is assigned a small region [3] around the corresponding point in the feature space. Data augmentations such as cropping, resizing, vertical or horizontal flipping are all considered to be included within the region this data sample represents.

For convenience, we do not consider partially overlapping scenarios here, otherwise the model would be too complicated. Every time when a new data sample is added into the dataset, there are two possible outcomes, one is that the small region it represents is entirely outside the sampled data area, the other is that it belongs to the sampled area (as shown in Figure 3.2). Due to intrinsic similarities among real-world data, as the number of samples grows, it is highly possible that a newly added sample is a near-duplicate of existing samples, which means they would have smaller chances to enlarge the whole represented area and only increase the redundancy.

3.2.1 Effective Number

In [3], there is a concept of effective number defined as: The effective number of samples is the expected volume of samples or the number of unique prototypes. The mathematical definition of effective number is:

$$E_n = \frac{1-\beta^n}{1-\beta} \tag{3.1}$$

$$\beta = (N-1)/N \tag{3.2}$$

Therefore, the larger the region of each sample is, the smaller the N is. Using the induction method, the mathematical formulation of effective number with regard to sample number of a class could be derived easily. Suppose that $E_1 = 1$ is the initial condition. E_{n-1} is the expected volume of the previous sampled data when trying to sample the n^{th} one. p is the probability for the n^{th} sample overlaps with the former sampled data area, which could be estimated as E_{n-1}/N. Therefore, we get:

$$E_n = pE_{n-1} + (1-p)(E_{n-1} + 1) \tag{3.3}$$

Assume $E_{n-1} = (1-\beta^{n-1})/(1-\beta)$ holds, then by combining formula 3.1 and formula 3.3, we could find $E_n = (1-\beta^n)/(1-\beta)$ also holds. According to the paper, the hyperparameter β controls how fast the represented area grows as the number of sampled data increases. Furthermore, this could also be interpreted as the i^{th} sample contributes β^{i-1} to the effective number. When N is extremely large (infinity large), the effective number is exactly the number of samples, which means every sample is unique. On the other hand, when N is close to 0, all data samples are represented by a single prototype.

3.2.2 Class-wise Weights

The idea of effective numbers with regard to each class provides us the concept that the model's bias toward particular classes' representations is not exactly proportionate to the sample numbers in each class. Instead, we should use effective numbers to approximate the bias and do reweighting operations on class level. However, a fatal problem is that we could not get any information about how to define the parameter N. According to [3], in practice, N is assumed to depend only on dataset. This means N is identical for all classes in the same dataset, which also makes the parameter β to be determined for all classes. Furthermore, the reweighting factor was set simply to be the inversely proportional to the effective number of each class. The reweighting operation is implemented on the loss function with regard to each class as equation 3.4 and 3.5 shown below. The exponential factor of β is exactly the number of training samples in the dataset.

$$Loss = \frac{1}{Batch} \Sigma_{i \in Batch} Loss(pred, gt) * w_{y_i} \tag{3.4}$$

$$w_{y_i} = (1-\beta)/(1-\beta^{n_{y_i}}) \tag{3.5}$$

It is worth noticing that the class-wise reweighting term is independent to the choice of loss functions, which could be easily applied to different loss functions such as SoftMax Cross-entropy loss, Sigmoid Cross-entropy loss and Focal loss. Especially, the focal loss was originally proposed to address the imbalance problem of positive and negative predictions. The class-wise weights could be regarded as a special way of setting the a parameter in the original Focal loss.

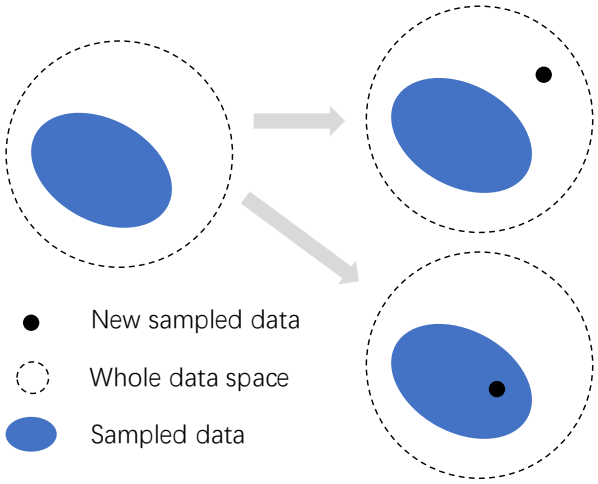

Figure 3.2 Relation Between the Sampled Data and the New Sampled Data [3]

3.3 Conditional Weights

Recall the conditional weighting factor introduced in former paragraphs, in the practical training process, we seek to reduce the error represented in equation 3.3. Now that we already have the class-wise weighting factor, the only thing left is the conditional weighting factor ε. The goal of the model is to be trained on an imbalanced dataset, and then perform reasonable predictions on balanced dataset (we assume in real scenarios the testing data distribution is more balanced). The goal could be represented mathematically as follows [2]:

$$\min_{\varepsilon} \frac{1}{|D|} \sum_{i \in D} L\big(f\big(x_i; \theta^*(\varepsilon)\big), y_i\big) \tag{3.6}$$

$$with: \quad \theta^*(\varepsilon) = argmin_{\theta} \frac{1}{|T|} \sum_{i \in T} (w_{y_i} + \varepsilon_i) L(f(x_i; \theta), y_i) \tag{3.7}$$

Here, the D stands for a balanced dataset developed from the original dataset, which is simply done by extracting a fixed number of samples from each class. T here represents the remaining dataset after the curation. The reason why we use the remaining dataset instead of the original dataset is that if the model is trained on the original dataset and then fine-tuned on the balanced dataset, there could possibly be an obvious overfitting problem, because this is basically the same as training the model with partially repeated data time after time. $f(x_i; \theta)$ here is the model's prediction and y_i is the ground truth label correspondingly. Equation 3.6 shows our expectation on the model when doing prediction, but we must aware that this is not the testing procedure and is not using the testing dataset either.

The method for training a model with such ability requires numerous computation resources, which makes us extremely hard to do simple training on the network with different loss function. Here we used the method proposed in [2], implementing a two-

stage learning approach which involved with a modified version of meta-learning. The basic training strategy is to do pretraining with normal training strategy (traditional training loss) on dataset T for the first stage, and do fine-tuning on the balanced dataset on the second stage. The detailed algorithm [2] is shown below.

Algorithm 1. Two-stage Learning for Long-tailed Problem

1 **While epoch < epoch_num1 do:** **(Stage 1)**

2 **Loop** for each minibatch B from the training set T

3 Compute traditional training loss, $\mathbf{L_B = \frac{1}{|B|}\sum_{i\in B} L(f(x_i;\theta), y_i)}$

4 Update model parameters with gradient decent, $\mathbf{\theta = \theta - \eta \nabla_\theta L_B}$

5 **End While**

6 **While epoch < epoch_num2 do:** **(Stage 2)**

7 **Loop** for each minibatch B from the training set T

8 Set $\varepsilon_i = 0$ for all samples in B

9 Compute weighted training loss, $\mathbf{L_B = \frac{1}{|B|}\sum_{i\in B} (w_{y_i} + \varepsilon_i) L(f(x_i;\theta), y_i)}$

10 Update model parameters with gradient decent, $\mathbf{\tilde{\theta}(\varepsilon) = \theta - \eta \nabla_\theta L_B}$

11 Get a minibatch B_d from balanced set D

12 Compute traditional training loss, $\mathbf{L_{B_d} = \frac{1}{|B_d|}\sum_{i\in B_d} L\left(f\left(x_i;\tilde{\theta}(\varepsilon)\right), y_i\right)}$

13 Update the conditional weight, $\tilde{\varepsilon} = \varepsilon - \tau \nabla_\varepsilon L_{B_d}$

14 Compute weighted training loss with updated conditional weight,

$$\tilde{L}_B = \frac{1}{|B|}\sum_{i\in B} (w_{y_i} + \tilde{\varepsilon}_i) L(f(x_i;\theta), y_i)$$

15 Update model parameters, $\mathbf{\theta(\varepsilon) = \theta - \eta \nabla_\theta \tilde{L}_B}$

16 **End While**

In algorithm 1 above, **stage 1** and **stage 2** are separated by preset epoch numbers (i.e., **epoch_num1** and **epoch_num2**). The split of the datasets should be done by data preprocessing in advance. The model focuses on getting some general knowledge in stage 1 (line **1-5**) by training on the remaining dataset T with traditional training loss (e.g., cross-entropy) denoted as L, which is marked in purple. This gives the model the

ability to recognize the objects in majority classes. θ denotes the model parameters and η represents the learning rate during updates with the computed loss.

The second stage (line **6-16**) thereafter mainly aims to raise the ability of the model to do fair prediction on minority classes. Specifically, in line **7-10**, the model parameters are updated for one step on dataset T. The loss function used here (marked in red) is the proposed form with class-wise weights. The conditional weights are all initialized as 0 at the start of each loop after sampling a minibatch from the balanced dataset, so that even if the loss function uses the combined weighting factors, it is actually the same to the class-wise weight alone.

In line **11-13**, the algorithm shows the way of updating the conditional weights. The conditional weight ε could be regarded as the model's prediction bias on each minibatch, which caused by the imbalance problem in stage 1. The weights are adjusted only with the knowledge of each minibatch, where we could see that the update is actually realized in a greedy way. We hope the model to fit all data by fitting every sampled piece of the whole picture.

At the end of each loop, in line **14-16**, the model uses the updated conditional weights to further adjust the model parameters, which could affect the prediction of minibatch within the next loop.

3.4 Two-stage Training Strategy in Object Detection

As stated in the previous paragraphs, the method is mainly designed for classification tasks only. As a result, there exists a gap to implement the algorithm on object detections tasks. The main challenge is that for object detection, the input images are not interleaved objects, but a whole scenery that contains multiple objects with various background noises and overlaps between objects. In algorithm 1, the requirement of creating a balanced dataset could not be achieved by simple removing some of the images from the original dataset. To adapt the algorithm to object detection field, we are now introducing our design of a masking strategy.

In the object detection framework, localization and classification functions are realized independently, which means when we input the images into the algorithm, the model should already have the knowledge of the bounding box location. The aim of creating the developed datasets in algorithm 1 is just removing objects. Therefore, what we are doing here is to mask the selected bounding box with a black canvas on the corresponding images. Meanwhile, to reduce the effect of masked objects to its neighboring objects, the model detects the overlapping parts between bounding boxes and do recoveries if one of them is not selected.

With this design, we could get both the balanced dataset and the remaining dataset simultaneously without doing too much harm to the background information or the unrelated objects. Figure 3.3 and 3.4 below show a random image before the masking, after the remaining dataset after masking and the recovery. The same image for the

balanced dataset is shown below in Figure 3.5.

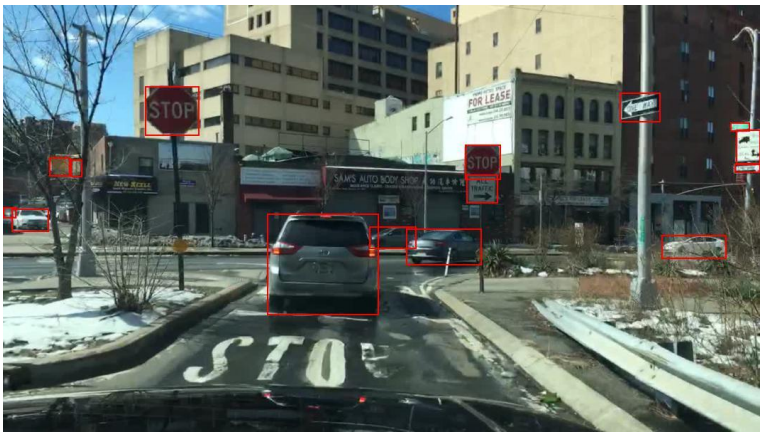

Figure 3.3 Original Image with Bounding Boxes Before Masking

Figure 3.4 Image for Remaining Dataset After Masking

Figure 3.5 Image for Balanced Dataset After Masking

Chapter 4 Experiments and Conclusions

4.1 Dataset

We use the BDD100K dataset for training and testing. Basically, BDD100K includes numerous images with 10 ground truth classes annotated with bounding boxes. The dataset consists of images taken under different times (such as day and night etc.), different backgrounds (such as residential, highway, city streets etc.), different weathers (such as clear, cloudy, foggy, rainy etc.).

The task is to do object detection in nighttime, so we only have access to the training samples during daytime. The testing samples are from both sides (i.e., daytime and nighttime). Figure 3.1 has already shown the data distribution of the training dataset and figure 1.1 also shows the long-tailed problem we are trying to deal with. Since that we choose to focus on the classification part, we cropped the objects from the original images and resized them (stretching and rescaling) to be 100*100 shape, so that the inputs are single objects with the same size, thus no need for localization. The following sections will show the experimental designs during this project, as well as the results of using algorithm 1 on image classification tasks. Last but not least, we have also tried to implement the strategy on object detection during nighttime, which will be illustrated during the last section.

4.2 Keep Rate

Before splitting out the balanced dataset, we should find a reasonable way to determine the size of the curation (we will call it meta-number in the following paragraphs). The largest class has more than 400k samples while the smallest class has only 95 (i.e., Train). If the meta-number is restricted by the sample number of the smallest class, the extracted samples are not guaranteed to well represent the corresponding classes with such limited amounts. We tried 200 and 2000 samples for meta-number settings for 15 epochs each (the second stage training requires nearly a week to do the whole 40 epochs). The results show that after 15 epochs of second stage training, for most of the classes, the accuracies are quite similar (differences within 5%), but fewer samples cost less time. Thus, we chose to set the meta-number as 200 samples to continue with further experiments.

By setting meta-number as 200 (larger than 95), we in fact gave up the smallest class and extracted 200 samples only from the remaining classes (there is actually no data sample for “Train” in the testing dataset). Till this end, the creation of the balanced dataset is completed with the following strategy:

$$
\begin{aligned}
&if\ n_i \geq 200: \\
&\quad keep\ rate = (n_i - 200)/n_i \\
&else: \\
&\quad keep\ rate = 0
\end{aligned}
$$

Here, n_i represents the sample number of class i, ranging from 95 to over 400k. The curation was done randomly with the keep rate calculated with the above strategy. Table 4.1 shows the sample numbers and the corresponding keep rates of each class.

Class ID	**1**	**2**	**3**	**4**	**5**
Categories	**Pedestrian**	**Rider**	**Car**	**Truck**	**Bus**
Sample Number	66882	3369	402892	20745	8559
Keep Rate	99.70%	94.01%	99.95%	99.03%	97.66%
Class ID	**6**	**7**	**8**	**9**	**10**
Categories	~~**Train**~~	**Motorcycle**	**Bicycle**	**Traffic Light**	**Traffic Sign**
Sample Number	~~95~~	2155	5168	90130	132502
Keep Rate	~~0%~~	90.72%	96.13%	99.78%	99.85%

Table 4.1 Sample Numbers and Keep Rates in Remaining Dataset

4.3 Experimental Details and Settings

Some detailed settings are shown in table 4.2 below. The first stage training lasts for 160 epochs with the second stage set as 40 epochs thereafter. The learning rate was set as 0.1 through the first training stage, whereas decays twice (at epoch 160 and 180 respectively) during the second stage. We expect the model to converge as the training goes during the fine-tuning stage. Parameter β is the one used to compute the effective number for each class as shown in equation 3.1 and 3.2. This is just a rough approximation (β should be less than but exceedingly close to 1), because the practice computation is actually unavailable.

Parameters	**Settings**
Classification Network Backbone	ResNet32
Training Batch Size	32
Number of Classes	10
Number for Meta-training (Meta-number)	200
Testing Batch Size	100
Epoch Number (Stage 1)	160
Epoch Number (Stage 2)	40

Baseline learning Rate (epoch 0-160) η	0.1
Meta-learning Rate (epoch 160-180) τ_1	0.001
Meta-learning Rate (epoch 180-200) τ_2	0.00001
Random Seed	42
β for Effective Number	0.9999
GPU	RTX 2080 Ti

Table 4.2 Detailed Experiment Settings

4.4 Experiment Results and Analysis for Classification

At this step, we verified the effectiveness of the method through implementing the two-stage training strategy on classification tasks with the settings above. We did experiments (ablation study) with baseline training strategy for nighttime testing dataset for comparison. In ideal case, after using the new training strategy, the accuracy for majority classes could drop within a reasonable range and the accuracy for minority classes should take a huge leap. The ultimate goal is to have similar high accuracy (i.e., over 90%) on all the class, or in other words, to lift the minorities to the level of the majorities.

4.4.1 Results

Table 4.3 below shows the results of the baseline classification which used the training strategy of the first stage for 200 epochs on nighttime images as the baseline. Then, table 4.4 shows the testing results for classification of daytime and nighttime objects after the 200 epochs of two-stage training strategy. The original training strategy took us around 3 days to get the results, whereas the 2-stage training strategy required nearly a week

Class ID	**1**	**2**	**3**	**4**	**5**
Categories	**Pedestrian**	**Rider**	**Car**	**Truck**	**Bus**
Test Night	90.06%	1.12%	92.46%	22.52%	22.22%
Class ID	**6**	**7**	**8**	**9**	**10**
Categories	~~**Train**~~	**Motorcycle**	**Bicycle**	**Traffic Light**	**Traffic Sign**
Test Night	~~0.0%~~	0.0%	41.6%	93.34%	89.48%

Table 4.3 Results for Baseline Training

Class ID	1	2	3	4	5
Categories	Pedestrian	Rider	Car	Truck	Bus
Test Day	85.75%	0.0%	96.58%	31.36%	37.32%
Test Night	80.01%	0.0%	93.14%	14.9%	16.67%
Class ID	6	7	8	9	10
Categories	~~Train~~	Motorcycle	Bicycle	Traffic Light	Traffic Sign
Test Day	~~0.0%~~	0.0%	53.61%	90.13%	90.21%
Test Night	~~0.0%~~	0.0%	35.11%	93.14%	85.61%

Table 4.4 Results for Two-stage Training

4.4.2 Problem Analysis

The method we used does not make any assumptions to the data distribution within the testing dataset. Thus, no matter what kind of distribution the nighttime data follows, the algorithm should all works. From table 4.4, it is obvious the testing results for the daytime dataset is better than that for the nighttime dataset. However, comparing to the baseline testing dataset in table 4.3, we found that there was in fact a performance drop. We analyzed the possible reasons for the bad performance and listed two possibilities below.

Firstly, the dataset we used to do classification is by cropping and resizing the labeled object from BDD100K. As stated in section 4.1, the objects experienced stretching or compressing to fit the input size, which might cause the model to learn wrong or modified features. Secondly, the resolution of the image was very low. Comparing to the dataset used in [2] (CIFAR10), what we used during the experiment had a much lower resolution that in some cases human could not tell what it is in the image. With the two reasons listed above, although the algorithm did not perform well during the image classification experiment, there is still a possibility that it works for original dataset BDD100K on object detection task.

4.5 Experiment Results and Analysis for Object Detection

To check the performance of the two-stage training strategy for object detection in practice, we did the following experiments on the whole images from BDD100K. The way we did curation to fit the data into our strategy was explained in section 3.4. The model was trained for 8000 epochs with normal training strategy, then trained with the two-stage strategy (only in classification part) for 2000 epochs (i.e., last 1/5 epochs of the training). The training data was the dataset from which we cropped the images to do the classification experiments in previous sections. The testing data is images during

nighttime.

4.5.1 Results

Tabel 4.5 and Figure 4.2 here show the result for object detection using two-stage training strategy in AP (average precision). Before explaining the "AP", we first introduce two concepts here, i.e., Precision and Recall. Denote TP as true positive prediction, FP as false positive prediction, FN as false negative prediction. These are used to measure how many samples are predicted and furthermore how many are predicted correctly. Then there are:

$$Precision = \frac{TP}{TP+FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP+FN} \tag{4.2}$$

The value in AP is calculated by an integral of P(r), which is the curve of Precision and Recall (as shown in equation 4.3). This is a very popular way of measuring the performance of deep learning model in object detection tasks, because it actually combines the measurement of accuracy and the hit rate of the bounding boxes.

$$AP = \int_0^1 P(r)dr \tag{4.3}$$

Class ID	**1**	**2**	**3**	**4**	**5**
Categories	**Pedestrian**	**Rider**	**Car**	**Truck**	**Bus**
AP Original	17.26	9.133	36.58	27.82	29.53
AP 2 stage	16.54	9.799	32.51	26.49	30.2
Class ID	**6**	**7**	**8**	**9**	**10**
Categories	**~~Train~~**	**Motorcycle**	**Bicycle**	**Traffic Light**	**Traffic Sign**
AP Original	~~0~~	8.345	12.87	8.874	24.87
AP 2 stage	~~0~~	7.476	12.18	9.349	23.28

Table 4.5 Results in AP for Object Detection

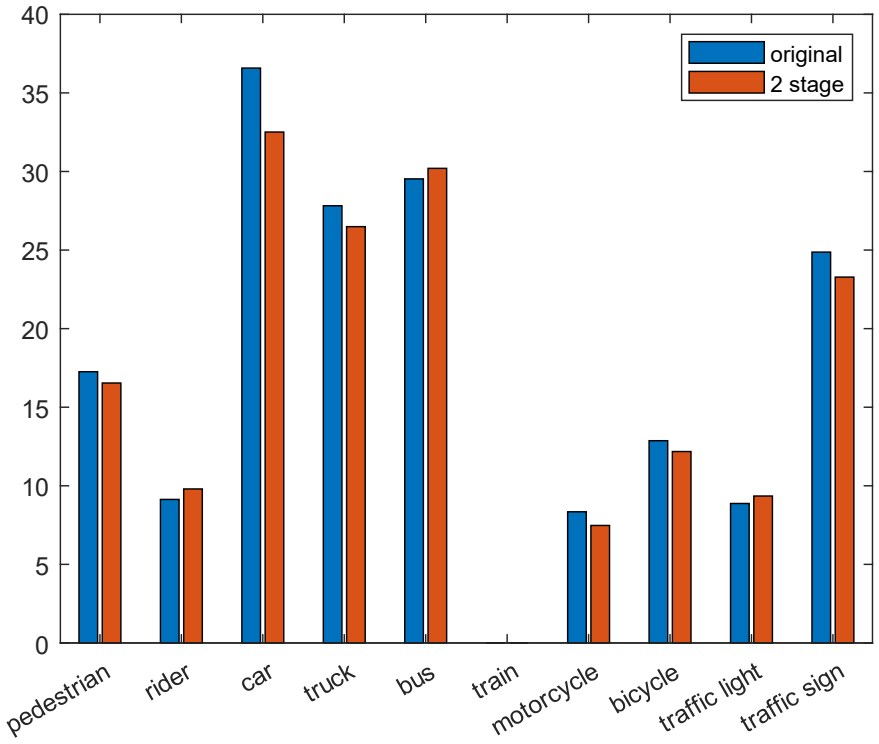

Figure 4.2 Results in AP for Object Detection

4.5.2 Problem Analysis and Solution

As we could see from the results, the difference between the two approaches was actually limited. Moreover, for most classes, the performance even decreased a bit. The reason for this could be two-fold. On one hand, for the nighttime data, the data distribution is also imbalanced. Therefore, what we expected from the beginning was that the performance would rise for the minority classes, with some acceptable decrease of the performance with respect to the majority classes. However, the truth is only the performance of three minority classes were improved (i.e., rider, bus and traffic light) but what we expected was about 5 to 6 classes (number of minority classes) should be improved. Thus, this is not the only cause of the problem.

Another reason could be that for object detection tasks, such problem might not only be caused by the classification part, but also the localization part. Here, we have two assumptions. One is that the predicted bounding boxes did not locate the objects with high IoU (i.e., the localization is not precise), with which the classifier could not produce good classification results either with the new training strategy. The other assumption is that the curation we did in section 3.4 makes the model learn incomplete or wrong features during training. Recall that we used a black canvas to cover the objects we selected to create the balanced dataset. When the curation is done, we actually removed not only the object itself, but also the background information of the image (i.e., the removed objects are also the backgrounds with respect to other objects). It is possible that the model would tend to believe that the background consists of black boxes and city environments during the training stage.

The most ideal way is to recover the background which was block by the removed object, but this is clearly impossible. We proposed two solutions to address this problem. One possible solution is to use a canvas with noises to cover the object. The distribution of the noise could be either Gaussian or a random mixed up of the pixels of the removed part. The logic behind is to approximate the environment (the blocked background) with noise, but in fact we could never know the exact distribution. The other solution is to mask the object with neighboring backgrounds (if any) or similar parts from other images. The idea behind this solution is that for city landscapes, the backgrounds are generally quite similar (i.e., buildings). However, there is still a chance that the background does not fit the environment. For example, if the mask of a car in the middle of the road happens to be a tree, the consistency of the background information could not be maintained.

Chapter 5 Conclusion

In this project, we implemented a two-stage training strategy to solve the class-imbalanced problem in image classification or object detection. The main idea is to combine the class-wise weight with the sample-wise conditional weight. It is worth noticing that we did not implement any changes within original loss function. Thus, the weights could be implemented together with any other loss functions, for instance, using focal loss with the computed weights might be an applicable way to improve the performance.

As for the following studies, experiments with the solutions stated in 4.5.2 as well as new losses might be a good direction to further improve the performance of the model in the near future. For the long-term plan, figuring out a way to improve the quality of the localization tasks should also be taken into consideration. Apart from these, we should also think about whether there is a way to combine the learning of domain-invariant features (specifically illumination invariant features) and resampling strategies to deal with such long-tailed problem.

References

[1] Oksuz, K., Cam, B.C., Kalkan, S. and Akbas, E., 2020. Imbalance problems in object detection: A review. IEEE transactions on pattern analysis and machine intelligence, 43(10), pp.3388-3415.

[2] Jamal, M.A., Brown, M., Yang, M.H., Wang, L. and Gong, B., 2020. Rethinking class-balanced methods for long-tailed visual recognition from a domain adaptation perspective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 7610-7619).

[3] Cui, Y., Jia, M., Lin, T.Y., Song, Y. and Belongie, S., 2019. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 9268-9277).

[4] Chen, Y., Li, W., Sakaridis, C., Dai, D. and Van Gool, L., 2018. Domain adaptive faster r-cnn for object detection in the wild. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3339-3348).

[5] Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28.

[6] Park, S., Lim, J., Jeon, Y. and Choi, J.Y., 2021. Influence-balanced loss for imbalanced visual classification. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 735-744).

[7] Yu, F., Wang, D., Chen, Y., Karianakis, N., Shen, T., Yu, P., Lymberopoulos, D., Lu, S., Shi, W. and Chen, X., 2022. SC-UDA: Style and content gaps aware unsupervised domain adaptation for object detection. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (pp. 382-391).

[8] Lin, T.Y., Goyal, P., Girshick, R., He, K. and Dollár, P., 2017. Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision (pp. 2980-2988).

[9] Wu, A., Han, Y., Zhu, L. and Yang, Y., 2021. Universal-prototype enhancing for few-shot object detection. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 9567-9576).

[10] Lengyel, A., Garg, S., Milford, M. and van Gemert, J.C., 2021. Zero-shot day-night domain adaptation with a physics prior. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 4399-4409).

[11] Wu, A., Han, Y., Zhu, L. and Yang, Y., 2021. Instance-invariant domain adaptive object detection via progressive disentanglement. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(8), pp.4178-4193.