Counting Pedestrians via Webcams*

Yafei Yan and Ruikun Li

RWTH Aachen University, Ahornstrasse 55, 52074 Aachen, Germany
http://cssh.rwth-aachen.de/

Abstract. Object detection is an challenged task since machine cannot observe objects like human who can instantly recognize an object. Based on the technique of convolutional neural network (CNN) the object detection became easy. By applying the object detection here we focus on counting pedestrians from a online or offline webcam.

Keywords: Counting Persons · Counting Pedestrians · Object detection via webcams

1 Introduction

1.1 Overview

This document depicts the practical project so-called "counting pedestrians in an image" that was done in the group of CSSH of RWTH University. The goal of this project is to find out all people frame by frame from an online webcam. In the project, the frame rate is defined as one screenshot per X seconds, where X is configurable. In other words, each X seconds a screenshot of the viewport of an online webcam is taken, then a screenshot, in which the number of persons and the bounding box of each detected persons is given, is output.

1.2 Motivation

Computer vision is now not a novel but still an interesting domain. When we are talking about computer vision, we first tend to recall image classification that is a base task of computer vision. However, the more interesting and complicated tasks are probably face detection, object tracking, object detection and so forth in our life. The breakthrough in computer vision is that in 2012 the AlexNet consisting of CNN (Convolutional Neural Network) won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge), which achieved a top 5 error rate of 15.4% for purpose of image classification, comparing to the next candidate with an error rate of 26.2%. A top 5 error rate is the rate that none of the top 5 labels matches the target label. From that time, almost all visual applications are developed based on CNN, even more advanced deforms of CNN were devised, which enables us to build a network achieving a decent performance, in

* Supported by organization x.

the case of our project, we compare the performance of person detection given by different state-of-the-art algorithms. It is good to know, how good an algorithm of person detection can be, since in complicated environments the detection task should be harder, for instance, occlusion, in the nighttime. Besides, on the side of applications, detecting pedestrians could be applied in the autonomous driving which is a very popular domain and counting persons could be used to predict emergent situations by a rapidly increasing number of people, schedule sufficient facilities according to the number of persons if an emergency happens. What's more, counting persons could be used in the area of social science. For example, researchers want to know whether the number of people is correlated with the weather. In short, this project could be used in multiple areas.

2 Approach

2.1 Pipeline

This section shows the architecture of the person counting project.

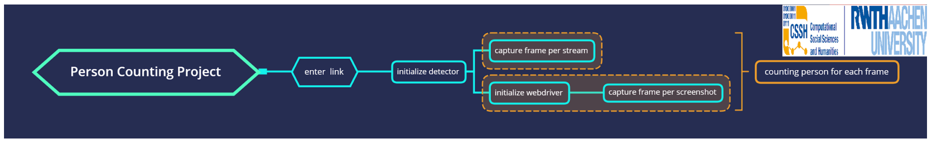


Fig. 1. The pipeline of this project, which also shows that the data flow.

Figure 1 shows the basic pipeline of the person counting project. The first step of the pipeline is to enter the stream link of a webcam. Then the detector should be initialized. In this project, we use a detector based on Python library ImageAI. When initializing the detector, the model(retina net, yolov3, yolo tiny), detect speed could be defined. After initializing the detector, there are two different methods to capture frames. The first one is capture frame per stream, it captures the frame simply from the stream link. The Python library Streamlink has been used. Another method is capture frame per screenshot, it applies the webdriver to open browser automation and takes screenshots and stores them as frames. The final step of the pipeline is counting person for each frame by the detector we initialized before.

3 Problem

During the development of this whole project we encountered a few problems that are worthy to be recorded.

3.1 Advertisements

When we visit a webcam via browser, Ads are shown before we have the view of the webcam. If we take a screenshot once we open the webcam, we get an unwanted image to be analyzed, thereby, we attach a line code of sleeping about 15 seconds to make sure we skip the ads.

3.2 Prerequisite for the way of screenshot

To count the pedestrians via opening a browser, taking screenshots, you must have a web browser and download a third-party browser webdriver from selenium. For webdrivers, we can pack webdrivers of top browsers to our project, but users still need to install a corresponding browser.

The browser may also need to provide the support of adobe flash, by which we are able to open a webcam by using the stream link without any problems.

3.3 The type of input images to the detector

For the screenshot method, there is only one input type of images to the detector, which is a real image. We take a screenshot first and then input it to the detector and detect how many pedestrians in the images. However, for the stream method, there are two possible input types. One is array, and the other is real image. The frame we captured from the stream is an array like image, therefore it can be directly inputted into the detector. Another input type is real image, we first capture the frame from the stream and use OpenCV to store it as a real image, after that input it into the detector. These two possible input types show different results. The following two figures 2, 3 show the difference between these two input types. Both figures apply resnet algorithm with normal speed. But figure 2(input type: real image) performs better than figure3(input type: array). With the first input type, the detector has detected 26 pedestrians, meanwhile with the second input type only has detected 17 pedestrians. This may due to the different RGB. (The second one has the blue background color.)

3.4 Variation of the results

The results of method capturing frame from the stream (input type: real image) are different to the results of algorithms comparison. The following two figures show the difference. Figure 4 is the result of method capturing frame from the stream (input type: real image), figure 5 is the result of algorithm comparison. Both of them apply resnet model and the same baseline. However, they have different detection results. We could find that the figure **??** not only detects more pedestrian but also has a higher percentage probability of detection. The reason for it is still not clear.

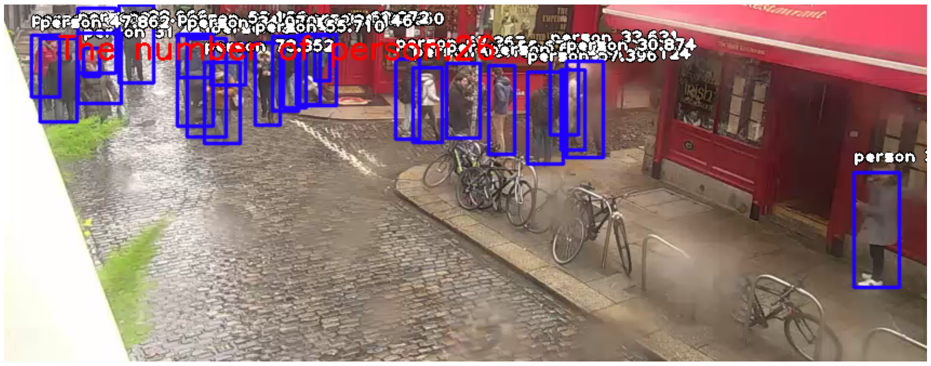


Fig. 2. Results of same image with input type real image

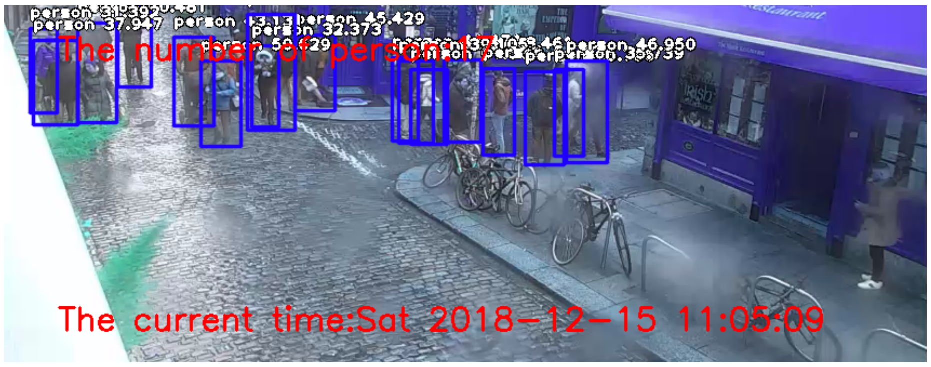


Fig. 3. Results of same image with input type array

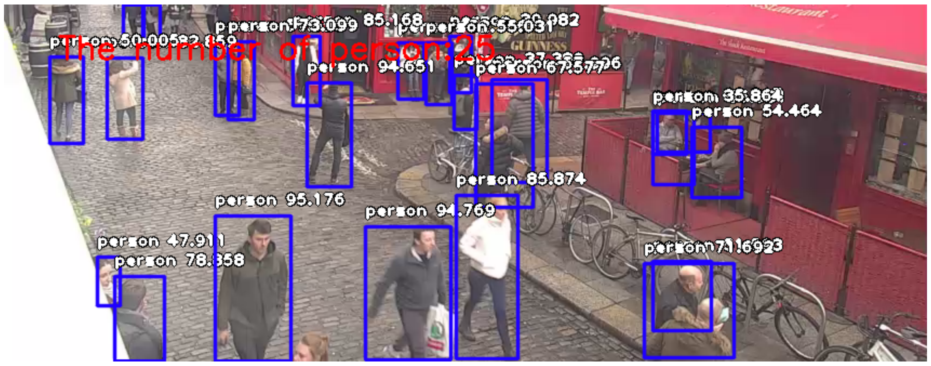


Fig. 4. result from the core method

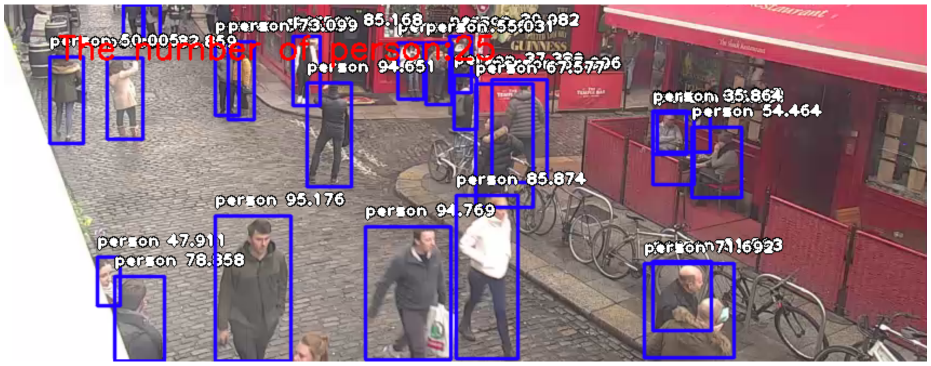


Fig. 5. result during methods comparison

3.5 Comparison of two methods

Although it seems method capture frame per stream is better than method capture frame per screenshot, the second method still gets some advantages in contrast to the first one. For example, the second method can only capture frames for stream link, which means it can only capture frames from a live video but not recorded videos or uploaded videos online. Counting pedestrian with recorded videos or uploaded videos online could be handled with the screenshot method.

4 Comparison

This section aims at explaining how the algorithms of pedestrian detection in our project work, by which we have an overview of the idea of them. By looking into these methods, we can figure out drawbacks and advantages of them, so that we are able to understand the evaluation result in the section of evaluation without effort. Since the project is based on the python library ImageAI which integrates three modern algorithms YOLOv3, Tiny YOLOv3, and RetinaNet. All of them are trained on COCO dataset.

Most of the traditional methods of object detection were devised to detect objects by sliding proposing template boxes at different scales from top left to bottom right of an image, for instance, the technique of sliding window. Intuitively, these techniques consume much effort on the progress of matching, since they traverse all available regions by sliding the window/template to do a match. Besides, objects probably occur in an image at any size so that the different sizes of sliding windows/templates must be given. In the following, we first introduce a new method with a different designing idea called YOLOv3.

4.1 YOLOv3

YOLO is short for You Only Look Once that directly indicates that you only look once at an image to detect objects and locate where they are. YOLO is a new approach different from other modern approaches of object detection because it reframes object detection as a regression problem by means of bounding boxes and associated class probabilities. The postfix "v3" indicates the version of YOLO which implies that there are some improvements in current version v3 compared to v1 and v2, but the basic framework does not have many changes, so here we present the base idea by means of the detail of YOLOv1. The YOLO detection system is quite simple, only 3 core steps shown in 6

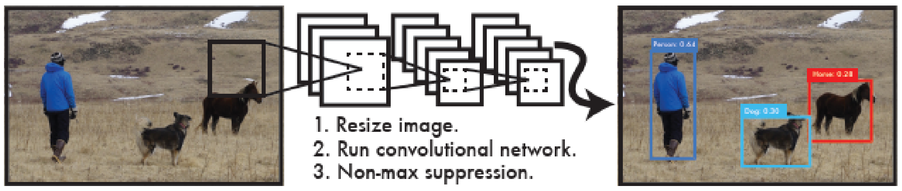


Fig. 6. The concise structure of YOLO detection system comprising 3 steps. Source: [1]

The main idea of YOLO is to divide an image into a 7x7 grid cells. For each grid cell, we predict M bounding boxes which are represented by a quintuple (x, y, w, h, con). The first two entries (x, y) point out the offset of the center of an object relative to the cell coordinate, (w, h) are the width and height of the predicted bounding boxes and con is the confidence defined as

$$Pr(object) * IOU_{pred}^{truth}$$

. If the center of the object falls in a cell, then

$$Pr(object)$$

= 1, otherwise 0. Then we compute the Intersection Over Union (IOU) of the ground truth and the predicted bounding box, so confidence is a fractional value shows how confident an object exists in that grid cell. Besides, each grid cell predicts C conditional class probabilities

$$Pr(class_i|object)$$

. Note all bounding boxes of a grid cell share the one copy of the C conditional class probabilities. By means of these parameters the model looks like below in the figure. So far we can infer that the class probabilities of a bounding box is computed as:

$$Pr(class_i|object) * Pr(object) * IOU_{pred}^{truth}$$

.

All above is a rough description of YOLO method and we omit many details of the network and training strategies and so on. In summary, this method has the following advantages.

– YOLO runs extremely fast, because the pipeline of the YOLO detection system is very simple like in the Figure. According to the evaluation result in the paper of YOLO v1 it is faster than other methods like Fast R-CNN.
– YOLO makes predictions by looking at the entire image instead of local regions so that it is prone to have less false positives that detect backgrounds as objects.
– YOLO is capable of learning generalizable representations, which means can detect different types of images even if we train the network on natural images and test it on artwork.

Of course, there are some limitations of YOLO. First, as mentioned in the beginning, YOLO tries to learn reasonable bounding boxes enclosing objects. If objects in a new or weird aspect ratio then the detection results become worse. Second, YOLO (v1) suffers from localization errors compared to other Fast R-CNN. Third, YOLOv1 gives two bounding boxes for each grid cell. As

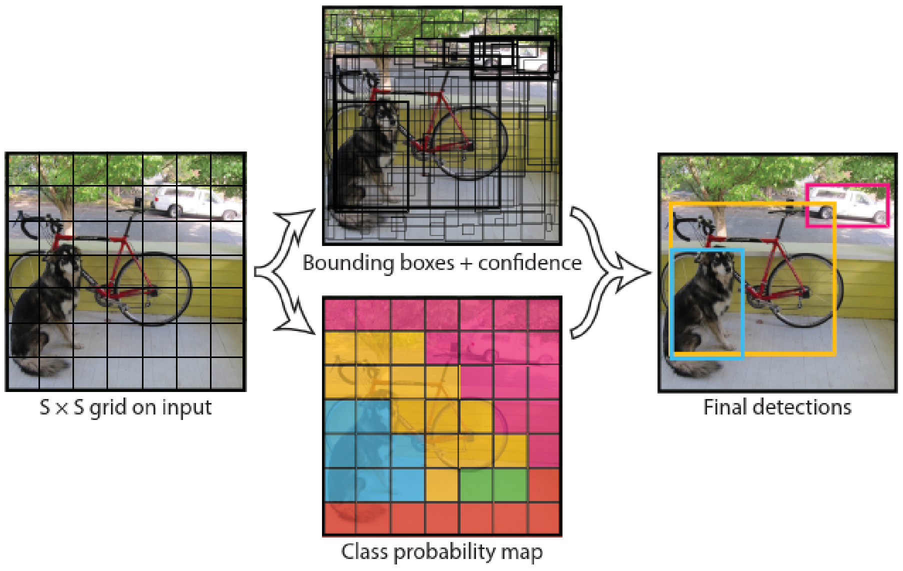
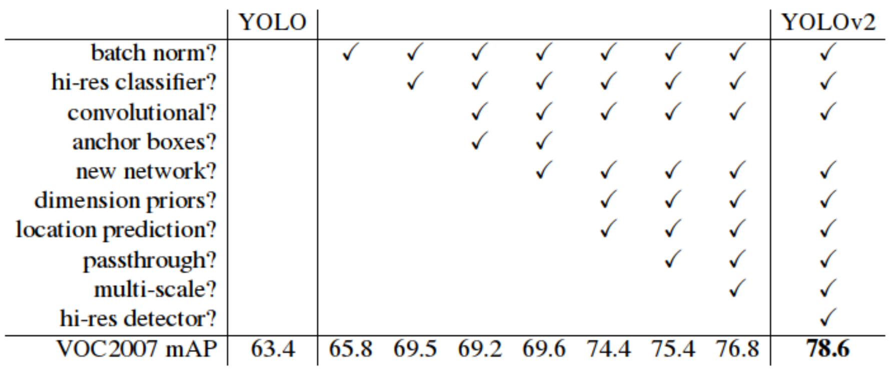


Fig. 7. YOLO Model. Dividing the image into an S x S grid cells and predicting B bounding boxes for each cell. Source: [1]

the YOLOv1 lag behind the Fast R-CNN at mean average precision (mAP) due to the disadvantages, the better version called YOLOv2 was issued by the researchers. They came up with a few strategies to improve the mAP by decreasing localization errors. The more specific descriptions are listed in the 8, in which we can directly observe the advancement. The mAP increases by attaching one op-



	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Fig. 8. The advancement of YOLOv2 compared to YOLO by attaching given strategies. source: [2]

timization strategy at each timestep. Nevertheless, the optimization strategy of anchor boxes does not give a positive gain in mAP, but in the paper it mentions that the recall (true positives over false+true positives) in this case increases. In the evolution progress of YOLOv2 to v3, like the authors wrote: just a bunch of small changes that make it better. The main small changes are residual network integrated and feature pyramid network (FPN) adopted. In addition, the number of convolutional layers increases at 53 compared to 19 of YOLOv2.

4.2 Tiny YOLOv3

Tiny YOLOv3 is a simplified version of YOLOv3 by cutting down some convolutional layers for extracting features. As if its prefix “Tiny” implies, it is lightweight, namely a tradeoff between runtime and precision. It could run faster than YOLOv3, but the precision would be a bit worse.

4.3 RetinaNet

The detectors which have the highest accuracy of object detection are two-stage detectors. The YOLO detector is a one-stage detector. Roughly speaking, the difference between two-stage and one-stage detectors is that the pipeline of two-stage detectors is more complicated, as the two-stage detectors invest much labor in extracting a sparse set of proposal regions at different scales, aspect ratios and locations in the first stage, by which classifies less number of regions in the second stage. The YOLO detector made a tradeoff between accuracy and speed. Although its performance is not one of the best, it is able to detect objects in real time in the case of ensuring a decent accuracy. However, the YOLO detector is unlikely to increase the accuracy even with a larger compute budget. In contrast, the goal of RetinaNet is to look up if the one-stage detector is able to outperform the accuracy of two-stage detectors when running at similar or faster speed. Obviously, RetinaNet is a one-stage detector, which is named for densely sampling of object locations in an input image.

In order to achieve that goal, RentinaNet introduces a novel loss function named as Focal Loss by adding two factors to standard cross entropy loss. The reason for the modification of loss function is that the problem of the large class imbalance occurs very often in classification tasks. The class imbalance is likely to overwhelm the cross-entropy loss for minimizing the loss during the training of an object detector, since the detector tends to stand on the side of the majorities that dominates the loss. In particular, we are often unable to collect positive training examples as many as negative training examples. On the contrary, in many situations the amount of positive examples is much less than the amount of negative examples, which leads to class imbalance especially if there is a dense detector. Obviously, if the proportion of positives and negatives is 1:99 and the standard cross entropy loss for binary classification is given as in the Equation $CE(p, y) = -log(p)$ if y =1, otherwise $-log(1-p)$ where $y \in \{+1, -1\}$ indicates ground-truth class and $p \in [0, 1]$ is the probability of predicting the class with label y = 1, to reduce the error the detector can simply predict all input examples as negative (non-object) then achieving the accuracy of 99% result in an amazing perfect detector. Nevertheless, that detector is quite poor. The problem was that we treat all examples equally, even if they are not. RetinaNet proposes to modify the loss function in two steps.

- Attach a weighting factor $\alpha \in [0, 1]$, for positive class and $1 - \alpha$ for negative class. In practice α might be configured by inverse class frequency, similar as inverse document frequency of TF-IDF. The more the class labels occur, the less the weighting factor is.
- Although α balances the importance of positive and negative examples, it does not take the easy and hard examples into account. Intuitively, the harder an example to be classified, the more important the example is. According to this idea, RetinaNet attached a modulating factor $(1 - p_t)^\gamma$ to the cross-entropy loss where $\gamma \geq 0$ and $p_t = p$ if y = 1, otherwise $1 - p$. What's more, the focal loss is visualized with different values of γ in the 9.

So far we present how RetinaNet solves the problem of class imbalance by means of a novel focal loss function. Equally important part is that RetinaNet detector has a simple and clear pipeline of its networks. It comprises a backbone network and two task-specific subnetworks. The backbone network takes efforts to compute a convolutional feature map over an entire image. The backbone's output, namely a convolutional feature map, is fed into the first task-specific

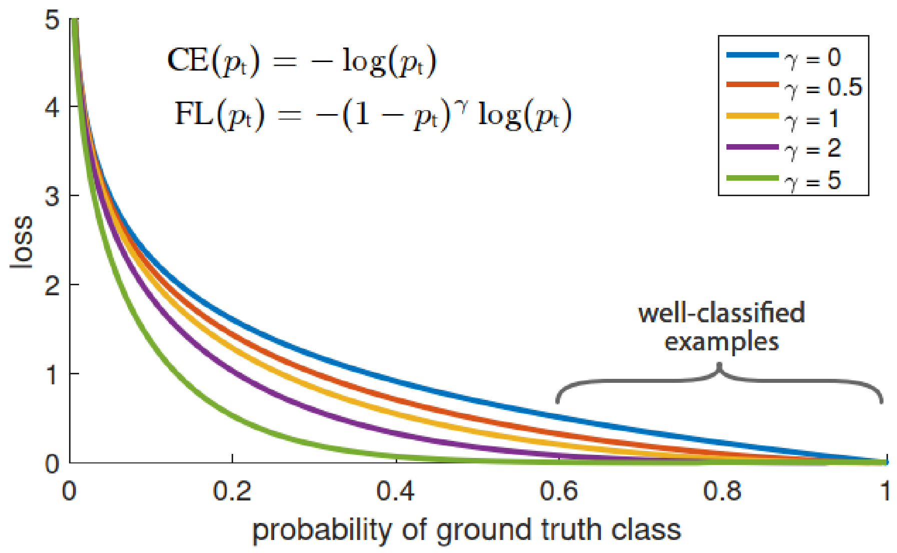


Fig. 9. $\gamma = 0$ implies the standard cross entropy loss which is represented by the blue curve. With positive γ the focal loss is reduced. The green curve with the highest γ value shows that the focal loss is reduced relatively large for hard examples ($p_t \leq 0.4$). source: [4]

subnetwork which is responsible for performing convolutional object classification. The second subnetwork performs convolutional bounding box regression. It is visualized in the 10.

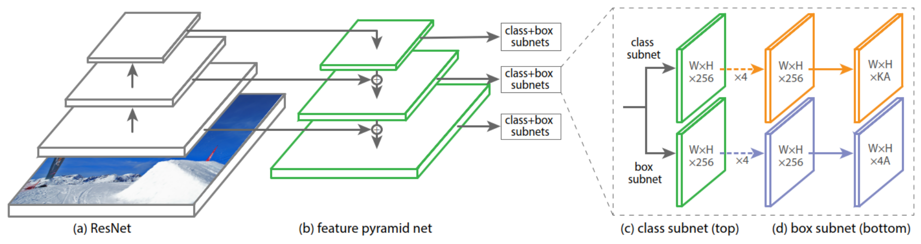
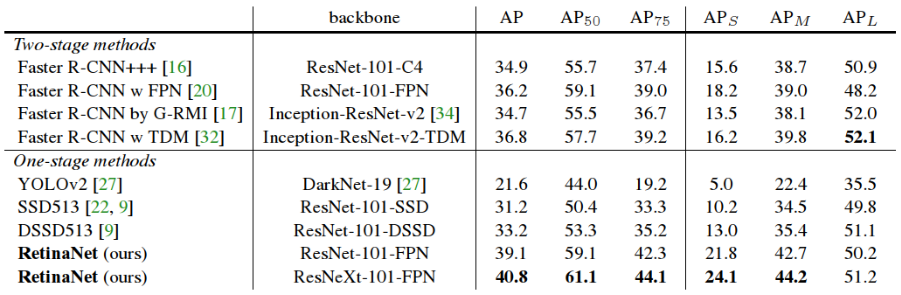


Fig. 10. (a) and (b): Backbone network is a Feature Pyramid Network (FPN) built on top of the ResNet (Residual Network) architecture. Go through the pathway of ResNet bottom-up to downscale getting feature maps at different scales. Applying skip connections (Residual) binds ResNet and FPN at each scale so that the FPN is able to upsample smoothly without losing more details. (c) and (d): subnetworks of classification on backbone's outputs and bounding box regression. source: [4]

In summary, the performance of RetinaNet can be proved by the experiment result in the 11.



	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2

Fig. 11. Evaluated on COCO test-dev. RetinaNet achieves top results and outperforms one-stage and two-stage detectors. Source: [4]

5 Evaluation

From the previous chapter, we go through the detail of three methods/algorithms that are applied in our project. In this chapter, the evaluation strategies and hypotheses and the result will be represented. We do not present the part of training as the three models pre-trained. Note that all scores results are rounded to keep three decimal numbers.

5.1 Test Dataset

All datasets and CSV files could be found in `https://drive.google.com/open?id=1ELg5LJoUOcoNjNZdjSXeaZLv6cI_jnKP`.

The detection results are stored in folder ***dublin_ day*** and ***dublin_ night***. The baseline images are stored in folder ***dublin_ day_ baseline*** and ***dublin_ night_ baseline***. The results of algorithms comparison are stored in folder ***algos_ dublin_ day*** and ***algos_ dublin_ night***. There is a CSV folder store all results of CSV files. The results of detection are stored in ***dublin_ day.csv*** and ***dublin_ night.csv***. Labeled results are stored in ***labeled_ dublin_ day.csv*** and ***labeled_ dublin_ night.csv***. The scores results included day and night are stored in ***dublin_ scores.csv***. For the algorithms comparison, there are two CSVs, namely ***Algos_ dublin_ day.csv*** and ***Algos_ dublin_ night.csv***, show the detection results of three algorithms (resnet, yolov3, yolo_tiny) from randomly select 15 images of the ***dublin_ day*** and ***dublin_ night*** dataset. And six CSVs for the labeled results of three algorithms and day and night. The scores result included day and night of three algorithms are stored in ***algo_ dublin_ scores.csv***

Intuitively, in the test phase the model can be tested on real data, so the test dataset is constructed by taking images from an online webcam and includes 100 images. We executed our python program to collect 50 images from a webcam of a walking street in Dublin in the daytime and nighttime separately. The webcam took images periodically, namely at every 3 minutes, to ensure an image distinguished from its neighbor images, because of people on that street flow by the webcam. After collecting these 100 images, we counted pedestrians in each image by ourselves and store the number in a CSV file. In order to count the number of pedestrians precisely, we also store the baseline datasets. Note that the counted number is inexact but approximately, since the scene of an image often contains many people who are very close to each other, resulting in occlusion. In addition, pedestrians were far from the webcam so that it was hard to recognize them that occurred in an image with a very small size. Thereby, we crop images and took the bottom half part of each image to ensure the test results are not in a quite tough situation. The following two figures 12, 13 show the conditions before cropped and after cropped. So the pedestrians in the alley could not be considered.

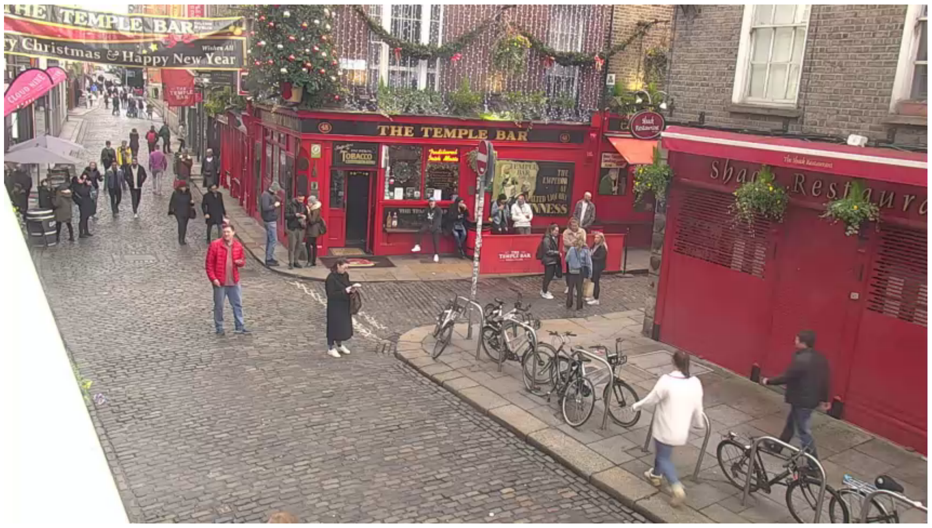


Fig. 12. original image

What is more, because of the counted number of images is inexact, it is necessary to make some subsidiary labels. For instance, if the counted number is greater than the median and the difference between the detected number and the counted number is smaller than the standard deviation of the greater part,

Fig. 13. cropped image

we assume they are the same, if the counted number is smaller than median and the difference between the detected number and the counted number is smaller than the standard deviation of the smaller part, we assume they are the same. We use these subsidiary labels to evaluate the dataset.

5.2 Strategies and Hypotheses

In order to evaluate these three pre-trained models properly, we make a few reasonable strategies. We randomly select 15 images from each baseline dataset(day and night) and apply them in three different algorithms, after that we use the same method to evaluate it.

Daytime and Nighttime In practice, the illumination in the daytime and in the nighttime has a huge difference and the change of illumination will influence the accuracy of object detection, even if the human is unable to have a good performance in the nighttime, especially the environment is very dark. We let each model perform on the dataset of daytime and the dataset of nighttime alternatively. Roughly speaking, we hypothesize that the test accuracy on the dataset of nighttime is unlikely to be as good as the accuracy on the dataset of daytime for each model.

The following table 1 and table 2 show the result of the detected number, time and the counted number in dataset day and night. From these two tables, we could find that the accuracy is not very high. Because of the counted number of images is inexact, it is necessary to make subsidiary label.

The following table 3 and table 4 show the result of the labeled dataset in day and night. We add one subsidiary label to help for the evaluation. **label_same** shows whether the counted number and detected number are assumed as the same. If the counted number is greater than the median and the difference between the detected number and the counted number is smaller than the standard deviation of the greater part, we assume they are the same, if the counted number is smaller than median and the difference between the detected number and the counted number is smaller than the standard deviation of the smaller part, we assume they are the same.

	image_name	detected_num	time	counted_num
0	dublin_day0.png	3	Fri 2018-12-28 10:15:08	4
1	dublin_day1.png	20	Fri 2018-12-28 10:18:18	23
2	dublin_day2.png	3	Fri 2018-12-28 10:21:23	6
3	dublin_day3.png	1	Fri 2018-12-28 10:24:29	2
4	dublin_day4.png	3	Fri 2018-12-28 10:27:35	2
5	dublin_day5.png	3	Fri 2018-12-28 10:30:40	4
6	dublin_day6.png	2	Fri 2018-12-28 10:33:46	3
7	dublin_day7.png	4	Fri 2018-12-28 10:36:52	4
8	dublin_day8.png	4	Fri 2018-12-28 10:39:57	4
9	dublin_day9.png	7	Fri 2018-12-28 10:43:03	8
10	dublin_day10.png	1	Fri 2018-12-28 10:46:09	2
11	dublin_day11.png	13	Fri 2018-12-28 10:49:15	15
12	dublin_day12.png	13	Fri 2018-12-28 10:52:21	16
13	dublin_day13.png	4	Fri 2018-12-28 10:55:27	4
14	dublin_day14.png	8	Fri 2018-12-28 10:58:32	6
15	dublin_day15.png	8	Fri 2018-12-28 11:01:38	9
16	dublin_day16.png	9	Fri 2018-12-28 11:04:44	17
17	dublin_day17.png	4	Fri 2018-12-28 11:07:51	13
18	dublin_day18.png	15	Fri 2018-12-28 11:10:57	16
19	dublin_day19.png	11	Fri 2018-12-28 11:14:03	15
20	dublin_day20.png	16	Fri 2018-12-28 11:17:09	14
21	dublin_day21.png	10	Fri 2018-12-28 11:20:15	15
22	dublin_day22.png	10	Fri 2018-12-28 11:23:22	16
23	dublin_day23.png	15	Fri 2018-12-28 11:26:28	15
24	dublin_day24.png	14	Fri 2018-12-28 11:29:33	16
25	dublin_day25.png	5	Fri 2018-12-28 11:32:39	7
26	dublin_day26.png	7	Fri 2018-12-28 11:35:45	6
27	dublin_day27.png	11	Fri 2018-12-28 11:38:51	15
28	dublin_day28.png	8	Fri 2018-12-28 11:41:56	13
29	dublin_day29.png	17	Fri 2018-12-28 11:45:02	17
30	dublin_day30.png	23	Fri 2018-12-28 11:48:08	30
31	dublin_day31.png	14	Fri 2018-12-28 11:51:14	14
32	dublin_day32.png	19	Fri 2018-12-28 11:54:20	23
33	dublin_day33.png	22	Fri 2018-12-28 11:57:28	21
34	dublin_day34.png	13	Fri 2018-12-28 12:00:34	19
35	dublin_day35.png	17	Fri 2018-12-28 12:03:40	39
36	dublin_day36.png	21	Fri 2018-12-28 12:06:45	43
37	dublin_day37.png	26	Fri 2018-12-28 12:09:51	52
38	dublin_day38.png	25	Fri 2018-12-28 12:12:57	65
39	dublin_day39.png	27	Fri 2018-12-28 12:16:02	55
40	dublin_day40.png	10	Fri 2018-12-28 12:19:08	24
41	dublin_day41.png	19	Fri 2018-12-28 12:22:14	23
42	dublin_day42.png	26	Fri 2018-12-28 12:25:20	35
43	dublin_day43.png	29	Fri 2018-12-28 12:28:26	40
44	dublin_day44.png	17	Fri 2018-12-28 12:31:32	25
45	dublin_day45.png	20	Fri 2018-12-28 12:34:38	36
46	dublin_day46.png	19	Fri 2018-12-28 12:37:44	27
47	dublin_day47.png	13	Fri 2018-12-28 12:40:49	18
48	dublin_day48.png	23	Fri 2018-12-28 12:43:56	27
49	dublin_day49.png	13	Fri 2018-12-28 12:47:02	15

Table 1. result of dublin day

	image_name	detected_num	time	counted_num
0	dublin_night0.png	19	Thu 2018-12-27 21:15:12	24
1	dublin_night1.png	18	Thu 2018-12-27 21:18:23	25
2	dublin_night2.png	21	Thu 2018-12-27 21:21:31	40
3	dublin_night3.png	23	Thu 2018-12-27 21:24:39	37
4	dublin_night4.png	29	Thu 2018-12-27 21:27:47	40
5	dublin_night5.png	22	Thu 2018-12-27 21:30:53	35
6	dublin_night6.png	17	Thu 2018-12-27 21:33:59	30
7	dublin_night7.png	17	Thu 2018-12-27 21:37:05	20
8	dublin_night8.png	21	Thu 2018-12-27 21:40:10	26
9	dublin_night9.png	15	Thu 2018-12-27 21:43:16	30
10	dublin_night10.png	19	Thu 2018-12-27 21:46:22	20
11	dublin_night11.png	17	Thu 2018-12-27 21:49:28	18
12	dublin_night12.png	25	Thu 2018-12-27 21:52:35	26
13	dublin_night13.png	17	Thu 2018-12-27 21:55:41	23
14	dublin_night14.png	16	Thu 2018-12-27 21:58:49	19
15	dublin_night15.png	10	Thu 2018-12-27 22:01:55	16
16	dublin_night16.png	15	Thu 2018-12-27 22:05:00	18
17	dublin_night17.png	19	Thu 2018-12-27 22:08:06	25
18	dublin_night18.png	10	Thu 2018-12-27 22:11:12	18
19	dublin_night19.png	18	Thu 2018-12-27 22:14:18	23
20	dublin_night20.png	22	Thu 2018-12-27 22:17:24	22
21	dublin_night21.png	17	Thu 2018-12-27 22:20:32	19
22	dublin_night22.png	16	Thu 2018-12-27 22:23:39	28
23	dublin_night23.png	24	Thu 2018-12-27 22:26:44	30
24	dublin_night24.png	23	Thu 2018-12-27 22:29:50	40
25	dublin_night25.png	24	Thu 2018-12-27 22:32:56	26
26	dublin_night26.png	17	Thu 2018-12-27 22:36:02	28
27	dublin_night27.png	20	Thu 2018-12-27 22:39:08	20
28	dublin_night28.png	18	Thu 2018-12-27 22:42:14	18
29	dublin_night29.png	22	Thu 2018-12-27 22:45:19	25
30	dublin_night30.png	19	Thu 2018-12-27 22:48:25	19
31	dublin_night31.png	17	Thu 2018-12-27 22:51:31	35
32	dublin_night32.png	17	Thu 2018-12-27 22:54:37	25
33	dublin_night33.png	19	Thu 2018-12-27 22:57:43	30
34	dublin_night34.png	20	Thu 2018-12-27 23:00:48	26
35	dublin_night35.png	21	Thu 2018-12-27 23:03:54	23
36	dublin_night36.png	19	Thu 2018-12-27 23:07:00	28
37	dublin_night37.png	20	Thu 2018-12-27 23:10:05	20
38	dublin_night38.png	22	Thu 2018-12-27 23:13:11	24
39	dublin_night39.png	13	Thu 2018-12-27 23:16:17	15
40	dublin_night40.png	13	Thu 2018-12-27 23:19:23	13
41	dublin_night41.png	14	Thu 2018-12-27 23:22:28	14
42	dublin_night42.png	14	Thu 2018-12-27 23:25:34	14
43	dublin_night43.png	7	Thu 2018-12-27 23:28:39	9
44	dublin_night44.png	14	Thu 2018-12-27 23:31:45	20
45	dublin_night45.png	10	Thu 2018-12-27 23:34:51	10
46	dublin_night46.png	14	Thu 2018-12-27 23:37:56	14
47	dublin_night47.png	10	Thu 2018-12-27 23:41:02	12
48	dublin_night48.png	11	Thu 2018-12-27 23:44:08	11
49	dublin_night49.png	16	Thu 2018-12-27 23:47:14	16

Table 2. result of dublin night

	image_name	detected_num	time	counted_num	label_same
0	dublin_day0.png	3	Fri 2018-12-28 10:15:08	4	same
1	dublin_day1.png	20	Fri 2018-12-28 10:18:18	23	same
2	dublin_day2.png	3	Fri 2018-12-28 10:21:23	6	not same
3	dublin_day3.png	1	Fri 2018-12-28 10:24:29	2	same
4	dublin_day4.png	3	Fri 2018-12-28 10:27:35	2	same
5	dublin_day5.png	3	Fri 2018-12-28 10:30:40	4	same
6	dublin_day6.png	2	Fri 2018-12-28 10:33:46	3	same
7	dublin_day7.png	4	Fri 2018-12-28 10:36:52	4	same
8	dublin_day8.png	4	Fri 2018-12-28 10:39:57	4	same
9	dublin_day9.png	7	Fri 2018-12-28 10:43:03	8	same
10	dublin_day10.png	1	Fri 2018-12-28 10:46:09	2	same
11	dublin_day11.png	13	Fri 2018-12-28 10:49:15	15	same
12	dublin_day12.png	13	Fri 2018-12-28 10:52:21	16	same
13	dublin_day13.png	4	Fri 2018-12-28 10:55:27	4	same
14	dublin_day14.png	8	Fri 2018-12-28 10:58:32	6	same
15	dublin_day15.png	8	Fri 2018-12-28 11:01:38	9	same
16	dublin_day16.png	9	Fri 2018-12-28 11:04:44	17	same
17	dublin_day17.png	4	Fri 2018-12-28 11:07:51	13	not same
18	dublin_day18.png	15	Fri 2018-12-28 11:10:57	16	same
19	dublin_day19.png	11	Fri 2018-12-28 11:14:03	15	not same
20	dublin_day20.png	16	Fri 2018-12-28 11:17:09	14	same
21	dublin_day21.png	10	Fri 2018-12-28 11:20:15	15	not same
22	dublin_day22.png	10	Fri 2018-12-28 11:23:22	16	same
23	dublin_day23.png	15	Fri 2018-12-28 11:26:28	15	same
24	dublin_day24.png	14	Fri 2018-12-28 11:29:33	16	same
25	dublin_day25.png	5	Fri 2018-12-28 11:32:39	7	same
26	dublin_day26.png	7	Fri 2018-12-28 11:35:45	6	same
27	dublin_day27.png	11	Fri 2018-12-28 11:38:51	15	not same
28	dublin_day28.png	8	Fri 2018-12-28 11:41:56	13	not same
29	dublin_day29.png	17	Fri 2018-12-28 11:45:02	17	same
30	dublin_day30.png	23	Fri 2018-12-28 11:48:08	30	same
31	dublin_day31.png	14	Fri 2018-12-28 11:51:14	14	same
32	dublin_day32.png	19	Fri 2018-12-28 11:54:20	23	same
33	dublin_day33.png	22	Fri 2018-12-28 11:57:28	21	same
34	dublin_day34.png	13	Fri 2018-12-28 12:00:34	19	same
35	dublin_day35.png	17	Fri 2018-12-28 12:03:40	39	not same
36	dublin_day36.png	21	Fri 2018-12-28 12:06:45	43	not same
37	dublin_day37.png	26	Fri 2018-12-28 12:09:51	52	not same
38	dublin_day38.png	25	Fri 2018-12-28 12:12:57	65	not same
39	dublin_day39.png	27	Fri 2018-12-28 12:16:02	55	not same
40	dublin_day40.png	10	Fri 2018-12-28 12:19:08	24	not same
41	dublin_day41.png	19	Fri 2018-12-28 12:22:14	23	same
42	dublin_day42.png	26	Fri 2018-12-28 12:25:20	35	same
43	dublin_day43.png	29	Fri 2018-12-28 12:28:26	40	not same
44	dublin_day44.png	17	Fri 2018-12-28 12:31:32	25	same
45	dublin_day45.png	20	Fri 2018-12-28 12:34:38	36	not same
46	dublin_day46.png	19	Fri 2018-12-28 12:37:44	27	same
47	dublin_day47.png	13	Fri 2018-12-28 12:40:49	18	same
48	dublin_day48.png	23	Fri 2018-12-28 12:43:56	27	same
49	dublin_day49.png	13	Fri 2018-12-28 12:47:02	15	same

Table 3. labeled result of dublin day

	image_name	detected_num	time	counted_num	label_same
0	dublin_night0.png	19	Thu 2018-12-27 21:15:12	24	same
1	dublin_night1.png	18	Thu 2018-12-27 21:18:23	25	not same
2	dublin_night2.png	21	Thu 2018-12-27 21:21:31	40	not same
3	dublin_night3.png	23	Thu 2018-12-27 21:24:39	37	not same
4	dublin_night4.png	29	Thu 2018-12-27 21:27:47	40	not same
5	dublin_night5.png	22	Thu 2018-12-27 21:30:53	35	not same
6	dublin_night6.png	17	Thu 2018-12-27 21:33:59	30	not same
7	dublin_night7.png	17	Thu 2018-12-27 21:37:05	20	not same
8	dublin_night8.png	21	Thu 2018-12-27 21:40:10	26	same
9	dublin_night9.png	15	Thu 2018-12-27 21:43:16	30	not same
10	dublin_night10.png	19	Thu 2018-12-27 21:46:22	20	same
11	dublin_night11.png	17	Thu 2018-12-27 21:49:28	18	same
12	dublin_night12.png	25	Thu 2018-12-27 21:52:35	26	same
13	dublin_night13.png	17	Thu 2018-12-27 21:55:41	23	not same
14	dublin_night14.png	16	Thu 2018-12-27 21:58:49	19	not same
15	dublin_night15.png	10	Thu 2018-12-27 22:01:55	16	not same
16	dublin_night16.png	15	Thu 2018-12-27 22:05:00	18	not same
17	dublin_night17.png	19	Thu 2018-12-27 22:08:06	25	not same
18	dublin_night18.png	10	Thu 2018-12-27 22:11:12	18	not same
19	dublin_night19.png	18	Thu 2018-12-27 22:14:18	23	same
20	dublin_night20.png	22	Thu 2018-12-27 22:17:24	22	same
21	dublin_night21.png	17	Thu 2018-12-27 22:20:32	19	same
22	dublin_night22.png	16	Thu 2018-12-27 22:23:39	28	not same
23	dublin_night23.png	24	Thu 2018-12-27 22:26:44	30	not same
24	dublin_night24.png	23	Thu 2018-12-27 22:29:50	40	not same
25	dublin_night25.png	24	Thu 2018-12-27 22:32:56	26	same
26	dublin_night26.png	17	Thu 2018-12-27 22:36:02	28	not same
27	dublin_night27.png	20	Thu 2018-12-27 22:39:08	20	same
28	dublin_night28.png	18	Thu 2018-12-27 22:42:14	18	same
29	dublin_night29.png	22	Thu 2018-12-27 22:45:19	25	same
30	dublin_night30.png	19	Thu 2018-12-27 22:48:25	19	same
31	dublin_night31.png	17	Thu 2018-12-27 22:51:31	35	not same
32	dublin_night32.png	17	Thu 2018-12-27 22:54:37	25	not same
33	dublin_night33.png	19	Thu 2018-12-27 22:57:43	30	not same
34	dublin_night34.png	20	Thu 2018-12-27 23:00:48	26	not same
35	dublin_night35.png	21	Thu 2018-12-27 23:03:54	23	same
36	dublin_night36.png	19	Thu 2018-12-27 23:07:00	28	not same
37	dublin_night37.png	20	Thu 2018-12-27 23:10:05	20	same
38	dublin_night38.png	22	Thu 2018-12-27 23:13:11	24	same
39	dublin_night39.png	13	Thu 2018-12-27 23:16:17	15	same
40	dublin_night40.png	13	Thu 2018-12-27 23:19:23	13	same
41	dublin_night41.png	14	Thu 2018-12-27 23:22:28	14	same
42	dublin_night42.png	14	Thu 2018-12-27 23:25:34	14	same
43	dublin_night43.png	7	Thu 2018-12-27 23:28:39	9	same
44	dublin_night44.png	14	Thu 2018-12-27 23:31:45	20	not same
45	dublin_night45.png	10	Thu 2018-12-27 23:34:51	10	same
46	dublin_night46.png	14	Thu 2018-12-27 23:37:56	14	same
47	dublin_night47.png	10	Thu 2018-12-27 23:41:02	12	same
48	dublin_night48.png	11	Thu 2018-12-27 23:44:08	11	same
49	dublin_night49.png	16	Thu 2018-12-27 23:47:14	16	same

Table 4. labeled result of dublin night

	df_name	accuracy	f1_score	precision_score	recall_score	r2_score	MAE	MAPE	MAD	accuracy_with_bool
0	labeled_dublin_night	0.24	0.24	0.24	0.24	0.05	5.32	0.195	3.0	0.52
0	labeled_dublin_day	0.12	0.12	0.12	0.12	0.481	6.14	0.276	3.0	0.72

Table 5. scores result of dublin

After making the subsidiary label, we could evaluate the dataset day and night. The table 5 shows the accuracy, f1 score, precision score, r2 score, mean absolute error, mean absolute percentage error, median absolute error and accuracy with bool. From this table, we could find that the mean absoulte error of night is 5.32 and of the day is 6.14. However, when it applies accuracy with bool, the accuracy of night stays but the accuracy of the day increases to 72%. In short, the detector performs better in the daytime.

Evaluation of algorithms We randomly select 15 images from each baseline dataset (day and night) and apply them in three different algorithms, after that we use the same method to evaluate it.

	df_name	accuracy	f1_score	precision_score	recall_score	r2_score	MAE	MAPE	MAD	accuracy_with_bool
0	lableled_resnet_dublin_night	0.067	0.067	0.067	0.067	0.697	3.8	0.16	2.0	0.6
0	lableled_yolov3_dublin_night	0.133	0.133	0.133	0.133	-0.399	8.533	0.288	5.0	0.2
0	lableled_yolot_dublin_night	0.0	0.0	0.0	0.0	-1.517	13.4	0.52	13.0	0.067
0	lableled_resnet_dublin_day	0.267	0.267	0.267	0.267	0.857	3.133	0.182	1.0	0.667
0	lableled_yolov3_dublin_day	0.267	0.267	0.267	0.267	0.472	6.2	0.277	1.0	0.6
0	lableled_yolot_dublin_day	0.0	0.0	0.0	0.0	0.222	8.667	0.545	3.0	0.333

Table 6. scores result for three algorithms of dublin

This table 6 shows the scores of three different algorithms in day and night. In the night time, the mean absolute error of resnet, yolov3 and yolo_tiny is 3.8, 8.533, 13.4 . And the accuracy with bool of them is 60%, 20%, and 6.7%. Obviously, resnet is the best algorithm in the night. What is more, yolov3 performs better than yolo_tiny in the night. In the daytime, the mean absolute error of resnet, yolov3 and yolo_tiny is 3.133, 6.2, 8.667. And the accuracy with bool of them is 67%, 60%, and 33%. Resnet is the best algorithm in the daytime without doubt regarding these scores. Meanwhile, yolov3 performs better than yolo_tiny in the daytime.

In short, resnet is the best algorithm no matter it is daytime or night time. And all algorithms perform better in the daytime.

References

1. Author, Joseph Redmon and Santosh Kumar Divvala and Ross B. Girshick and Ali Farhadi: You Only Look Once: Unified, Real-Time Object Detection. Journal **CoRR**, (2015)
2. Author, Joseph Redmon and Ali Farhadi: YOLO9000 Better, Faster, Stronger, Journal **CoRR**, (2016)

3. Author, Joseph Redmon and Ali Farhadi: YOLOv3: An Incremental Improvement, Journal **CoRR**, (2018)
4. Author, Tsung-Yi Lin and Priya Goyal and Ross B. Girshick and Kaiming He and Piotr D.: Focal Loss for Dense Object Detection, Journal **CoRR**, (2017)
5. ImageAI Github Homepage, `https://github.com/OlafenwaMoses/ImageAI`. Last accessed 04.01.2019