AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA TELEKOMUNIKACJI

PROJEKT DYPLOMOWY

Visual Bicycle Counter

Wizualne liczydło rowerowe

Autor: Szymon Wojciech Lenart
Kierunek studiów: Teleinformatyka
Typ studiów: Stacjonarne
Opiekun pracy: Dr hab. inż. Mikołaj Leszczuk, Profesor AGH

Kraków, 2020

Serdecznie dziękuję mojej Żonie, za okazane mi wsparcie w trakcie pisania tej pracy.

Contents

1. Introduction

Bicycle counter is a device that automatically counts cyclists riding on the road. Roadside counting devices for riding cyclists are installed in many cities, also in Krakow. They are usually part of the city's broader strategy, wanting to encourage its residents to ride bicycles. The bicycle counter usually works on induction loops embedded in a nearby bicycle path that detect passing bikes. Sometimes photocells are also used to count cyclists. However, this work's scope is to create visual bicycle counters, based on techniques for recognizing images from publicly available webcams. These data should be counted over a longer time horizon, and, optimally, be combined with data on cards from automatic, urban cycling measurement points.

1.1. Goals

My thesis's goal was to create a mechanism for counting cyclist on the video file, downloaded from publicly available street cameras' view. That, in turn, could help evaluate investments made by city authorities or plan future projects.

1.2. My Contribution

To achieve the mentioned goal, I had to research the newest Machine Learning and Computer Vision algorithms and techniques and learn how to customize their functioning. As an addition, I have not only trained the AI model to detect and count cyclist but also used it on more video files to get data and show good visualization and its interpretation.

1.3. Order of content

First of all, I will write about theoretical aspects of my work with *State of the Art* of Computer Vision and Machine Learning in traffic planning and analysis. Then I will describe the practical side of work I had to do to achieve the goals of my thesis. In the end, I would like to show the results of my work as well as the visualization of data my program collects with the exemplary interpretation of this data.

2. Theoretical aspects

In this chapter, I would like to describe a theory standing behind my work and its use nowadays.

2.1. State of the Art

Nowadays, broadly understood data becomes one of the most desired resources; everyone wants to research everything. For example, many mobile applications want their user to access localization or send some information to servers. However, the more data we have, the more time it requires from human to process it and extract only things we need. Top it all off it needs to be interpreted afterwards to make some sense of it. That is why Data Science becomes so popular these days. It tries to solve the problem of growing time taken to process bigger and bigger sets of data. The solution turned out to be simple - "let the machines do the work". That is how Machine Learning (ML) was born. Computer Vision (CV) in the other hand is an Artificial Intelligence (AI) branch specified for working with image/video, and it uses ML algorithms as a step to achieve its goals of extracting needed data from images/videos with as little human work as possible. As of today, CV finds more and more application in various science branches. The most known use of CV is the development of Autonomous Driving. There is also technology for monitoring and prevention of pests and grass diseases[1], based on CV. As an example, China's government works on the spread of street cameras with face recognition software, but that is less of a positive example as some would say, used for human behaviour control. However, in my knowledge, there is no Visual Bicycle Counter in a commercial application.

2.2. What a Bicycle Counter is and why make it "Visual"

As the name suggests, Bicycle Counter is a device for counting cyclists driving by a specific street or bike path. Usually, it is a convection loop embedded in the asphalt of a road, but photocells installed by the roadside are becoming more popular. Both devices count cyclists passing by and save the number on some server, to make it accessible to obtain for authorized people. That information can later serve as a good indicator of bicycle traffic in the city. In figure 2.1 we can see that there are 17 of those devices installed across the city and numbers from them are publicly available in Krakow.

Data	Armii Krajowej	Bora-Komorow-skiego	Bulwary	Dworzec	Grzegórzecka	Kamieńskiego	Klimeckiego	Kopernika	Kotlarska	Mogilska	Monte Cassino	Niepołomska	Nowohucka	Smoleńsk	Tyniecka	Wadowicka	Wielicka
2020/12/13	140	128	331	293	619	96	193	494	443	434	387	187	127	267	219	215	177
2020/12/12	127	140	316	344	656	152	215	532	513	533	397	136	132	319	185	259	198
2020/12/11	174	198	365	550	863	154	375	901	784	878	571	81	176	433	190	474	398
2020/12/10	143	218	413	561	737	152	406	895	776	890	535	94	166	420	220	404	425
2020/12/09	238	323	595	731	942	194	516	1121	1008	1230	740	173	223	573	301	527	570
2020/12/08	265	354	653	838	978		635	1262	1041	1391	818	223	251	559	416	703	640
2020/12/07	213	386	615	846	763		614	1271	1083	1380	793	118	272	588	388	661	644
2020/12/06		410	1668	596			700	1127	1017	1112	1075		358	446	1787	501	604
2020/12/05		329	1003	564			487	940	918	1029	807		304	484	774	536	462
2020/12/04		422	813	871			641	1382	1202	1494	819		282	619	454	666	635
2020/12/03		334	495	739			483	1111	997	1195	701		254	531	260	546	512
2020/12/02		336	504	762			486	1171	935	1133	671		209	571	258	512	526
2020/12/01		391	744	844			606	1330	1165	1450	822		325	583	459	668	654

Figure 2.1: First rows of data from bicycle counters installed in Krakow [2]

Devices like this will be more and more needed as more and more bicycles appear on streets. Using data from the oldest Bicycle Counter installed in Krakow (at Bulwary) we can see that number of cyclists grows slowly but surely in recent years (figure 2.2) what can alarm local government to make additional investments like new bike roads.
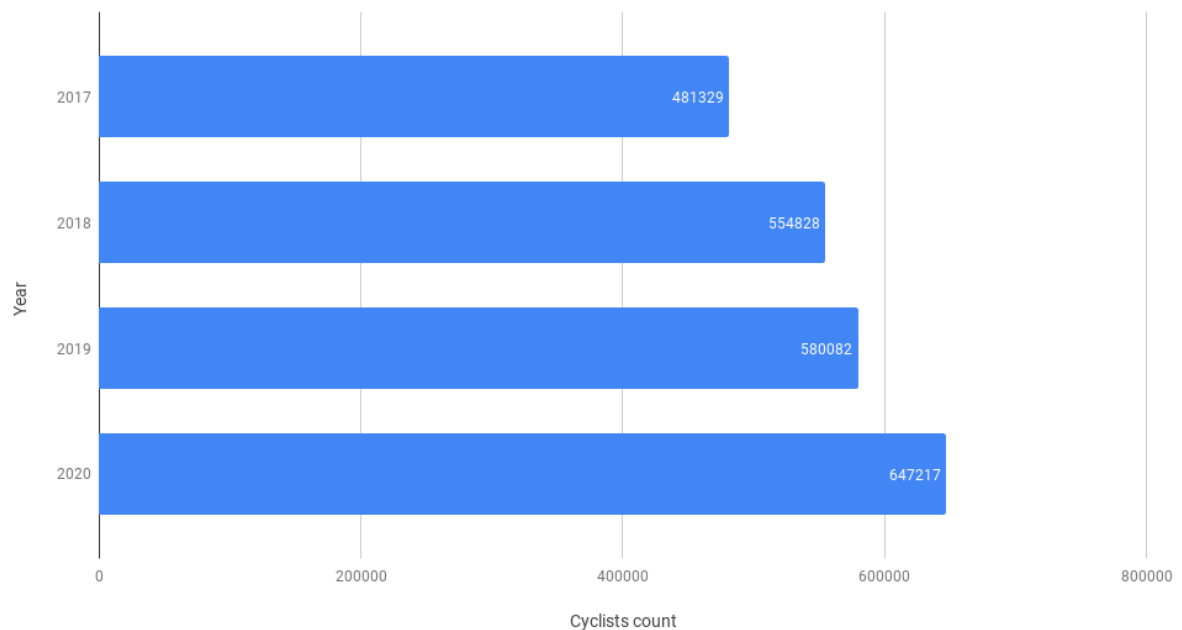

Figure 2.2: Cyclists count at Bulwary, Krakow

But why even bother about making Visual Bicycle Counter? First of all, devices I mentioned above are all other equipment used with only one purpose - counting cyclists, so their reusability leaves a lot to be desired. Secondly, installing such a device requires to put more work into it, because you need to, for example, make a hole in the road, put a convection loop in and patch the hole up. Same as maintenance can be problematic too. Visual Bicycle Counter could solve those issues. It uses a trained ML model to perform inference on video files from

street cameras downloaded on a computer or straight on video streams from websites that publicly share it. That would not add the next device but would use street cameras' existing infrastructure, widely developed in many cities.

2.3. Short theory behind Visual Bicycle Counter

Visual Bicycle Counter bases on deep learning model, what has an advantage over traditional target detection. "The traditional method is to manually extract features and require experts in related fields to manually design and process them through years of accumulation and experience. The method of deep learning can learn the features of difference in response data through a large amount of data and is more representative. The deep learning model simulates the human brain's visual perception system. It extracts features directly from the original image, and the features are passed through the layer by layer to obtain the high-dimensional information of the image, making it a great success in the field of computer vision."[3], But first I had to train my model of collecting data to create a dataset for training (in my case: label video frames - show precisely where the cyclist is, divide labelled frames to train and test). Training the model is "showing" computer training data, it learns those and based on what is learns it tries to predict desired information on test data and then the process is repeated many, many times until reaching satisfactory parameters. For me, the primary metrics were: Loss, Recall, Precision. Loss is a target function minimized in the optimization process (the lower, the better, and it gets lower when the model predicts more accurately). Recall means how many selected items should be selected, and Precision means how many selected items are our desired items. Visualization of Recall and Precision is shown in figure 2.3.
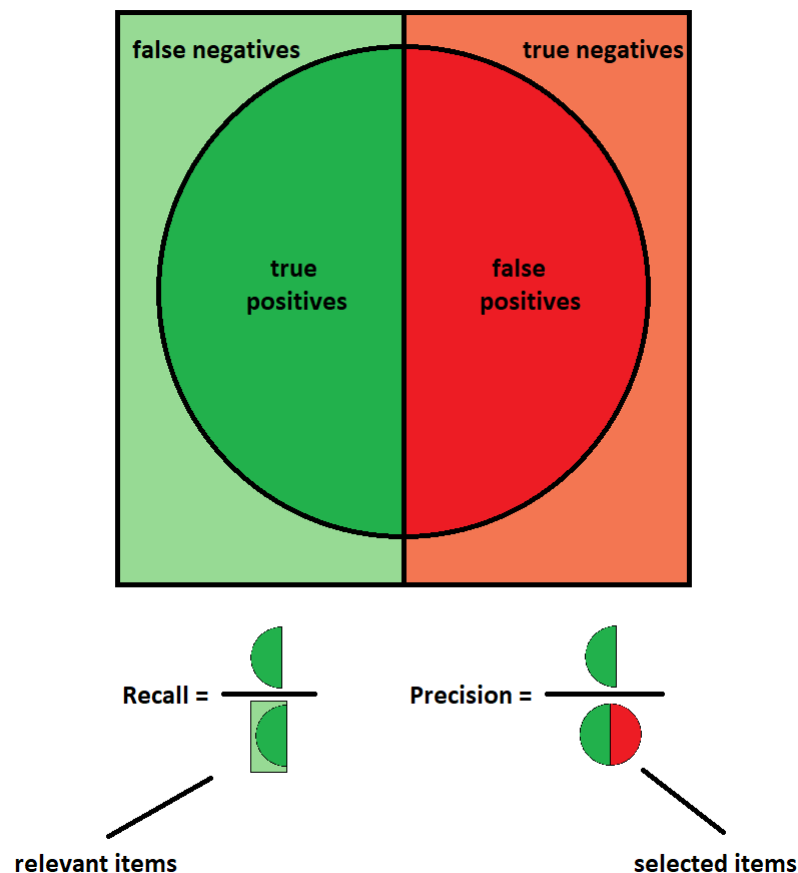

Figure 2.3: Recall and Precision

2.4. Tools selection

Tools I have used to create my Visual Bicycle Counter were as follow:

– youtube-dl software for downloading video streams from street cameras as mp4 files

– LabelImg program for manual image labelling

– Virtual Machine with Ubuntu 18.04.5 LTS for download scheduling

– Google Colaboratory as working environment,

– Python programming language,

– OpenCV libraries for object detection (used by YOLOv5 API),

– YOLOv5 API for training YOLO[4] models and performing inference,

– Wandb for training evaluation.

Most of the choices were pretty obvious. I was looking for widely used solutions that provide good documentation and a big user base. Youtube-dl was the easiest tool to install and use. Using Google Colaboratory (GC) machine was not only my choice but at a certain stage of work, I needed a more powerful GPU and CPU. GC provided all that as well as Python programming language support. The biggest challenge appeared when I was looking for Object Detection tools. I had to test different solutions, such as Tensorflow with Keras, but I found it not suitable for my use case, so I decided to use OpenCV libraries, YOlOv5 API and Wandb that appeared much more user friendly and provided all features I needed.

3. Practical aspects

To achieve my goal of creating Visual Bicycle Counter, I had to complete a few steps. With tools chosen, my work was to create a proper dataset for training ML model, train and evaluate this training to reach acceptable values of metrics I mentioned in section 2.3. After that, I could customize the API detection script to make it count cyclist rather than just detecting them using the trained model. Lastly, I used my edited script to run inference (detection) on other videos from a street camera on Lema Street, where bike road was modernized in July this year. I used videos before and after modernization, to prepare a simple comparison, as an example of how data obtained by my Visual Bicycle Counter can be used.

3.1. Data collection

First of all I had to collect data. To accomplish that, at the end of June 2020 I have created Virtual Machine (VM) with Ubuntu 18.04.5 Operating System (OS). Using Unix type OS was much easier and intuitive than using it on my Windows machine. After my VM was ready, steps to obtain video files from street camera stream were as follow:
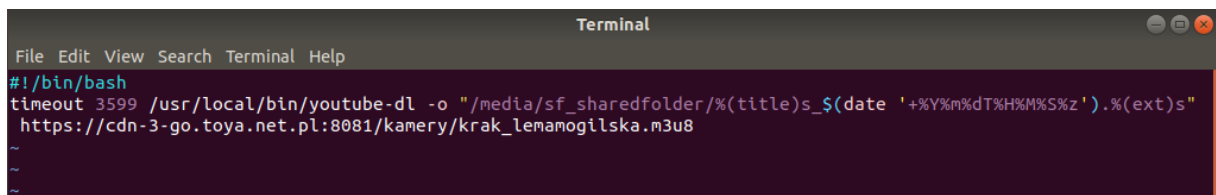
1. Installing ffmpeg for Ubuntu OS(from terminal):

    – sudo apt-get install ffmpeg

2. Downloading and installing youtube-dl (from terminal):

    – sudo curl -L https://yt-dl.org/downloads/latest/youtube-dl -o /usr/local/bin/youtube-dl
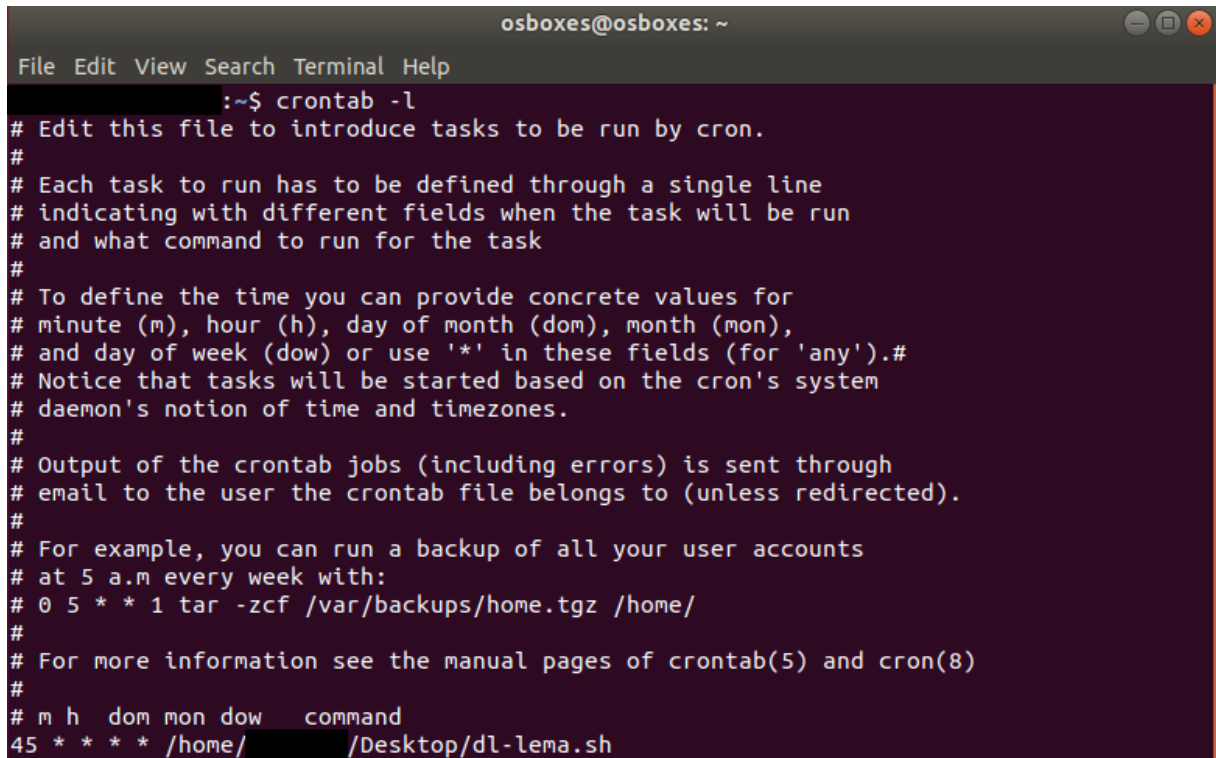
    – sudo chmod a+rx /usr/local/bin/youtube-dl

3. Creating bash script that runs youtube-dl with proper parameters what is shown in figure 3.1

Figure 3.1: This script runs youtube-dl, video files are saved to directory with name containing stream name on http website and date of download. After 3599 seconds youtube-dl gets terminated.

4. Adding created script to crontab -e to schedule its execution to every hour as we can see in figure below.

9

Figure 3.2: output of `crontab -l` command which shows what commands execution was scheduled

At this moment, downloading of video files proceeded automatically. Note that stable internet connection was very important as without it downloaded video files came out corrupted (damaged), what made them impossible to use later on.

3.2. Creating the dataset

After collecting enough videos I have started to create the dataset for training my YOLO model. This involved extracting frames from videos, choosing those with the most clear, visible cyclists, then manually labelling each image using LabelImg program and lastly divide labelled images on train and test data in ratio of approximately 8:2. To make it precise, those are steps I have taken:

1. Frame extraction with VLC media player

    – Scene Filter settings customization
       Tools -> Preferences -> Show settings -> All -> Video/Filters/Scene Filter
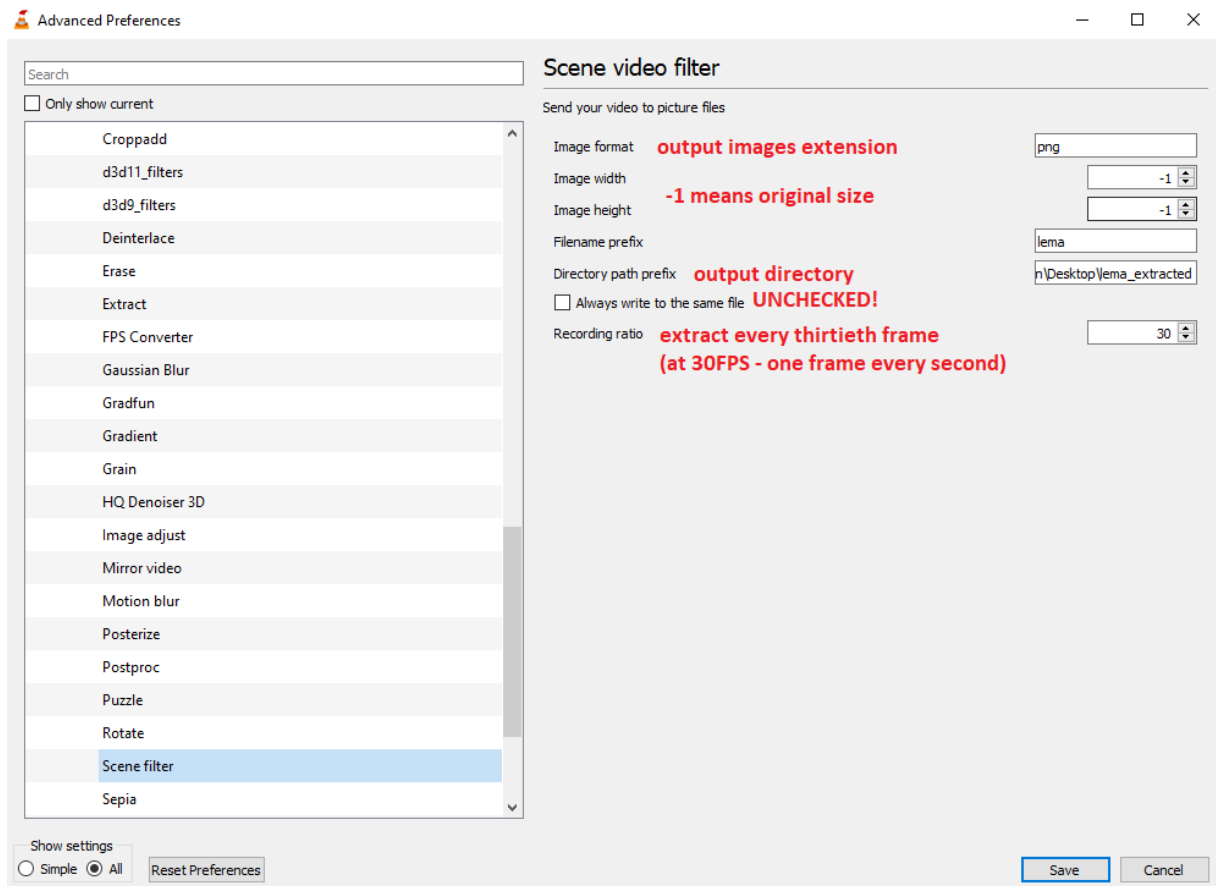
Figure 3.3: Scene Filter edit window

– Switching Scene filter on

> Tools -> Preferences -> Show settings -> All -> Video/Filters -> check Scene video filter

– Restart VLC and run video file with it (at this point frames are extracted automatically)

2. Selection of frames (I used about 200 different frames with various count of cyclists on them)

3. Manual frames labelling using LabelImg:

– after downloading LabelImg from repository, running it by executing command `python labelImg.py <images-path>` (Python3 or higher required) window shown in figure 3.4 appears. It shows labelled image as well. To create label we simply click Create RectBox

Figure 3.4: LabelImg user interface with labelled image
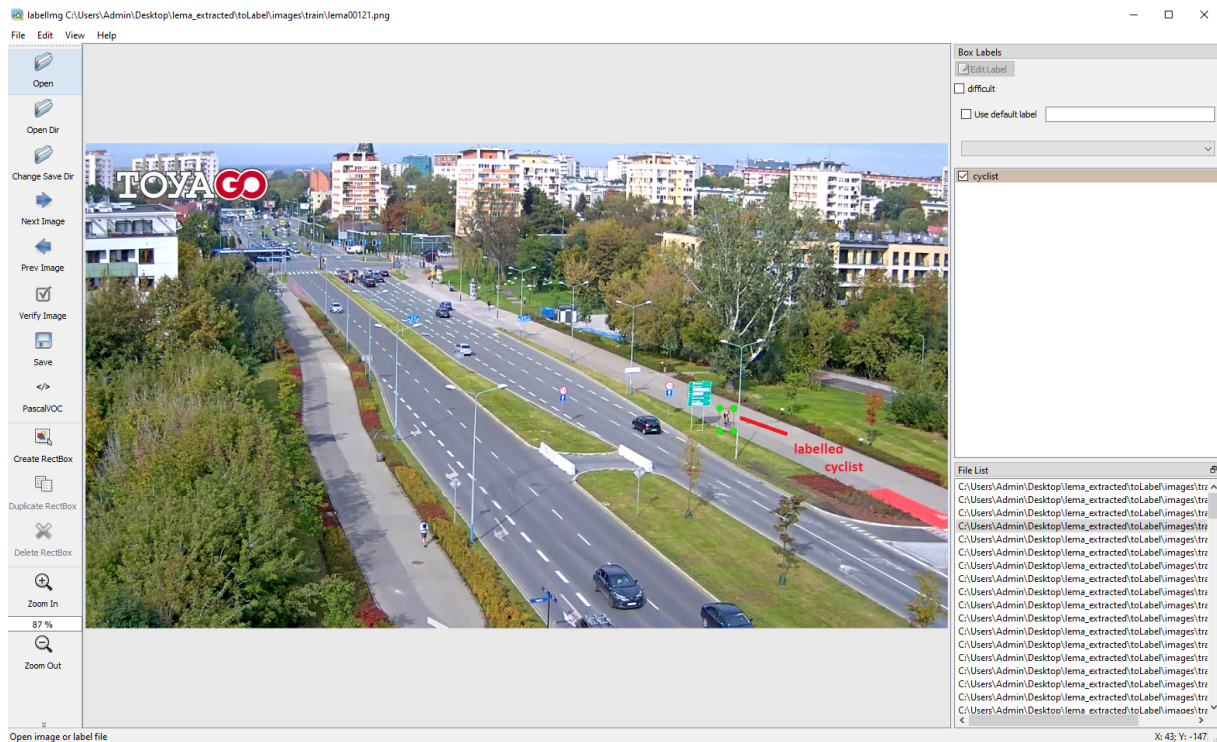
Saving labelled frame will generate text file for this image with numeric label representation (one line in file for each label on image).
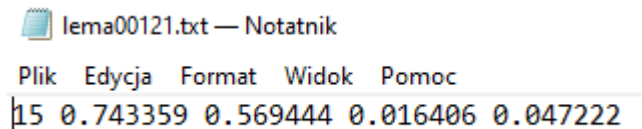

Figure 3.5: Text file generated for image in figure 3.4

4. Partition of images and corresponding text files on train and test (val) data. To accomplish and to do this easier I have written Python script which randomly divides images with text files to two directories "train" and "val" in desired ratio (in my case it was 80% to train the model and 20% to test it)

3.3. Environment preparation, training model and running detection

As my working environment, I have chosen Google Colaboratory (GC) machine. It provides me with three times more GPU memory and overall better (faster) GPU device than the one in my computer, resulting in approximately six times faster training and detection processes. It was essential while performing inference (detection) because GC's machine-made this operation execute close to real-time on 30 Frames Per Second (FPS) video files. My computer-processed five frames per second while GC's machine processed almost 30 frames per second, what saved much time. Also, it turned out much more effective while running inference straight on the video stream from the website. Lastly, GC's machine has good Python3 support what facilitated the work. I have created Jupyter Notebook file to run code cells one by one, to speed up my work, what came especially handy while starting work

on next day because sadly GC allows only a few hours of inactivity before restarting session and unsaved files are gone. To start training, I stored all necessary files like videos and labelled images on my Google Drive, connected to GC's machine. Also uploading files on the Drive and copying them on the machine was the fastest way to access my computer files. Training the model proceeded in few steps (figures below will show code cells that have been run):

1. After setting runtime type to GPU I mounted (connected) my Google Drive to GC's machine what creates a new directory on the machine named gdrive from which we can access Google Drive

```
[ ]  from google.colab import drive
     drive.mount('/content/gdrive')

     Mounted at /content/gdrive
```

Figure 3.6: Mounting Drive

2. Then I cloned YoloV5 GitHub repository

```
[ ]  !git clone https://github.com/ultralytics/yolov5  # clone repo
     !pip install -r yolov5/requirements.txt  # install dependencies
     %cd yolov5
```

Figure 3.7: Cloning repository to machine

3. After that, I could run the training process with installed wandb for evaluating the model during and after training.

```
[ ]  %pip install -q wandb
     !python train.py --img 1280 --batch 8 --epochs 300 --data dataset.yaml --weights yolov5l.pt
```

Figure 3.8: Command used for training the model

Parameters used in command:

– img - bigger of two numbers in video resolution

– batch - how many images to process at once (limited by GPU memory - in my case 8 was the biggest GPU could handle)

– epochs - how many times to train the model on whole dataset

– data - path to .yaml file which contains information about dataset

– weights - path to weights file - empty for training from scratch, I used pretrained weights of yolov5l model, which offers the best accuracy within my desired execution time (<40ms).

3.4. Training Evaluation

The first command from figure 3.8 installs wandb software on the machine. It automatically collects data during the training process and shows it in a more accessible manner - charts. Below we can see how parameters mentioned in section 2.3 (Recall, Precision and Loss) has been changing during the training process. Also, weights with the best combination of those three parameters (marked on charts) will be used for performing inference.
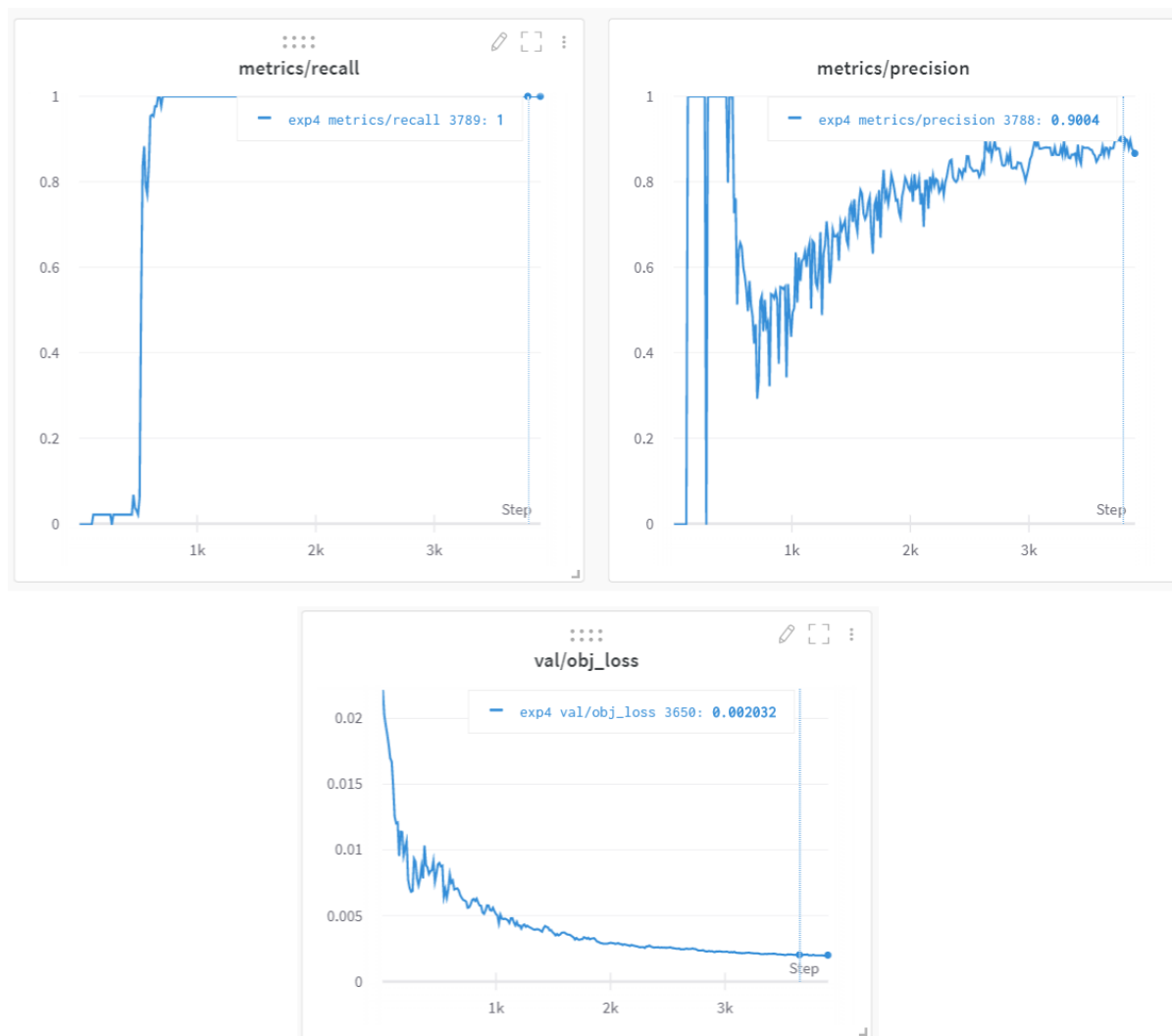
Figure 3.9: Changes of Recall, Precision and Loss during model training

As we can see, parameters of trained model I have been using while performing inference on video files I have collected look as follows:

Recall	Precision	Loss
1 (100%)	0.9004 (90%)	0.002032

What basically means that every cyclist on test images was found (Recall), and 90% of all detections are indeed cyclists (Precision). Also our model finds areas where detection is the most probable with very fast (Loss).

3.5. Counter implementation and running inference

Running detection process is also really easy with YOLOv5 API, because it contains ready to use script detect.py that can be executed right away on video file or stream. Implementing counter was more of a challenge, because it is not implemented in API. I have chosen straight forward solution - when a detected cyclist crosses a virtual line, counter value goes up. This approach was possible because of my Recall and Precision being on very good level. I knew every cyclist is detected on every frame of video (100% Recall) and 90% of detected objects are actually cyclists (Precision). Every script I have customized is put on public Github Repository[5]. Running detection process on GC's machine proceeded as follows:

1. Mounting Google Drive on GC's machine

```
[ ]  from google.colab import drive
     drive.mount('/content/gdrive')

     Mounted at /content/gdrive
```

Figure 3.10: Mounting Drive

2. Copying model from Drive, unzipping it and installing required dependencies

```
[ ]  !cp gdrive/MyDrive/dataset/Lema/models/yolo_trained_count_csv.zip .
     !unzip yolo_trained_count_csv.zip
     %cd yolov5/
     !pip install -r requirements.txt
```

Figure 3.11: Installing ready API on machine

3. Uploading video for perform inference on it (two ways)

```
[ ]  from google.colab import files
     uploaded = files.upload()
```

Figure 3.12: Uploading video file straight from local machine to GC's machine (slower)

```
[ ]    #add rest of path to video and uncomment next line
       #!cp ../gdrive/ .
```

Figure 3.13: Copying video file from mounted drive after uploading video file to it (faster)

4. Running detection process

```
[ ]  python detect.py --img-size 1280 --source krak_lemamogilska_20201027T094501-0400.mp4 --weights ./runs/train/exp4/weights/best.pt --classes 15 --conf-thres 0.4
```

Figure 3.14: Command used for performing inference on video file using customized `detect.py` script

Parameters used in command:

– img - bigger of two numbers in video resolution

– source - path to video file

– weights - path to weights file (`best.pt` are best weights from training model) basically it is path to model we will use to detect cyclist

– classes - identifier of detection classes to not filter out (`15` means that we only want to detect cyclists)

– conf - confidence threshold value (probability that detected object is indeed cyclist) - everything with higher confidence is considered as cyclist.

The figure below shows a frame from a video on which inference was performed (with Counter already implemented). During the testing phase, I have been saving every video generated during the inference process. However,

Figure 3.15: Frame from video file generated during the detection process

after that I quit doing that, because a generated video was almost five times bigger than the original one (1-hour video downloaded from website video stream was approximately 700MB big, so saving every result of inference would fill my Google Drive memory very quickly). Instead, I have implemented that at the end of execution of `detect.py` information about the video was written to `result.csv` file (video name, video duration, cyclist count).

3.6. Exemplary evaluation of results

To familiarize with the potential of Visual Bicycle Counter, I have prepared an exemplary interpretation of the Counter's data. I had chosen video files collected between 10 AM and 8 PM from a random week before modernization of bike road on Lema Street and performed inference on them. The same I have done with videos from the random week after modernization. I have aggregated this data to show mean value for each hour of the day of the week and put it on a graph. This is how the aggregated data looks like graphically:
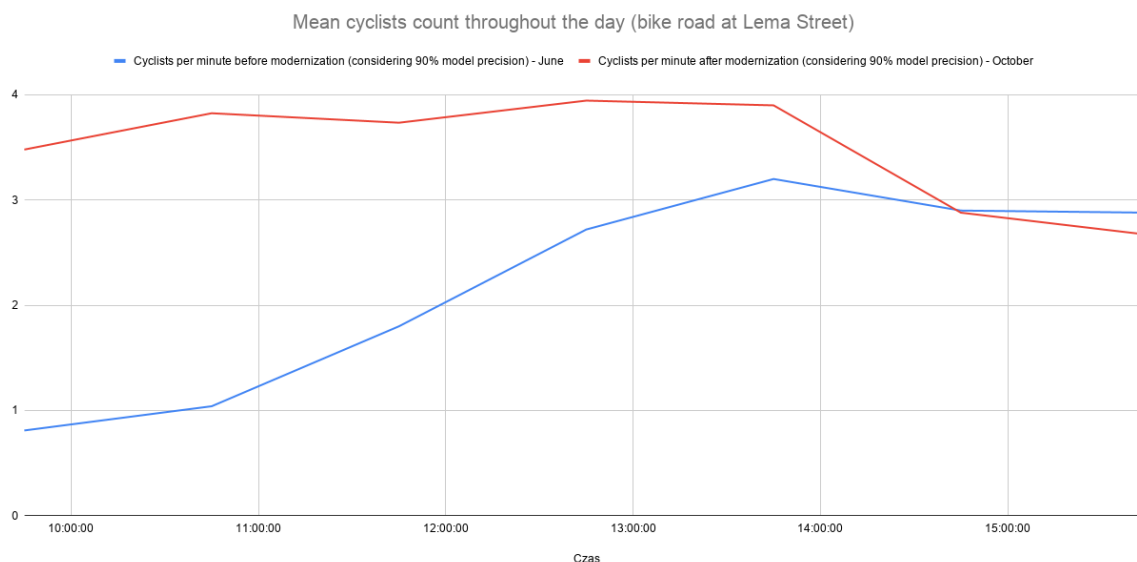

Figure 3.16: Mean cyclists count in week scale (Lema Street)

Even this simple example provides important information on whether investment turned out to be a success or not. We can see that even though data after modernization was collected in October - the month we cannot call cyclists favourite - number of cyclist per minute was higher than in June. It could be one of the ways to evaluate local government's projects. Second example showing different measure that can be taken with data collected by the Counter is to simply take daily cyclists count and compare it on the scale of the month. In figure 3.17 we can see cyclists count collected in June and September as well as sum of those numbers. Showing this data on a graph like in figure 3.18 won't show striking differences, but we can still see that overall more cyclists were detected in September than in June. There can be a lot of reasons why data is like this, but collected numbers are very good indicators to compare with other parameters like weather conditions in specific month (temperature, rainfall level).

| Day of the month	Cyclist count	
	June	September
1	2749	1016
2	2184	2273
3	2716	3158
4	2995	2738
5	1298	2788
6	3263	923
7	1894	1350
8	1579	3103
9	2177	3440
10	1876	2528
11	2385	2957
12	2053	2646
13	2117	2755
14	1819	3258
15	2676	3424
16	1967	3523
17	1674	2417
18	2044	2422
19	1572	2134
20	1752	2137
21	841	2795
22	1029	3241
23	3339	2951
24	2294	2831
25	2963	2036
26	2305	967
27	2197	1102
28	2751	1267
29	1601	1738
30	3545	726
Sum	65655	70644

Figure 3.17: June and September cyclists count
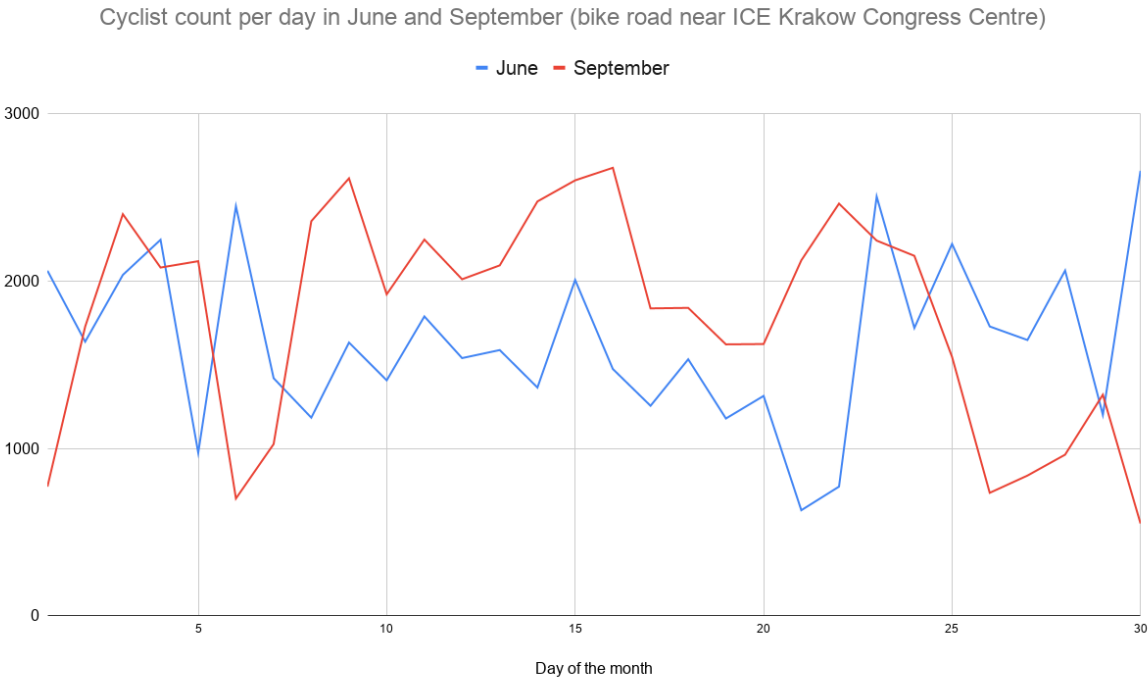
Figure 3.18: Cyclists count in June and September

4. Conclusions

To sum everything up, learning about Computer Vision entirely from scratch was a very illuminating experience. I had to go through theory to practice to explore the secrets of Machine Learning used in image processing. However, hours spent on various courses will not go in vain. I am working on this or different subjects because there is still plenty to do in this matter. Visual Bicycle Counter is only an introduction to traffic analysis or investments evaluation and planning, and it always takes more time spent on developing it. Different measures can also be taken using gathered data, not only those from section 3.6. The used model can also be trained to detect different objects like pedestrians, cars or electric scooters, getting more and more popular nowadays. It may be used to evaluate investments not only on bike infrastructure but also roads or sidewalks. As Machine Learning techniques are developed, more and more ways can be used and less human work it will require.

Bibliography

[1] Y. Li and Y. Zhang. Application research of computer vision technology in automation. In *2020 International Conference on Computer Information and Big Data Applications (CIBDA)*, pages 374–377, 2020.

[2] Mobilny kraków, http://mobilnykrakow.pl.

[3] W. Lan, J. Dang, Y. Wang, and S. Wang. Pedestrian detection based on yolo network model. In *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1547–1551, 2018.

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[5] https://github.com/miklesz/visual-bicycle-counters.