Estimating the distance between people on public spaces based on neural networks

Laszlo Makara, Botond Varsanyi, Gergely Safran

December 11, 2020

Contents

1 Introduction **4**

1.1 Detectron2 4

1.2 Datasets 4

1.3 Distance projection 5

1.4 Image slicing 6

2 Solution **7**

2.1 Train 7

2.2 Validation 8

3 Conclusion **9**

List of Figures

1.1 Detecting objects on image[1] 5
1.2 Roboflow tagged images 5
1.3 Generating image flows from video 6

2.1 Estameted distances 7
2.2 Train results 8
2.3 Validation results 8

Abstract

As Covid-19 spread across the globe social distancing became more and more important and also the wearing of masks. Because with these restrictions we can successfully decrease the daily new cases, and minimaze the fullness of hospitals. If we could monitor the distance between people in shops and other public places, then we could let them know with speakers if they are too close to each other. The goal is a system that could be used in inner and outer places as well, and can also operate at night time and can measure the distances between people.

Chapter 1

Introduction

Social distance calculating is though task on two dimensial images. There are a few possibilities in order to measure social distance. We tried to find a universal solution, meaning that it could esteem distance in pictures and videos as well. We could achieve that with slicing the videos into frames and process the frames. After that we teach our model with the frames. We chose the Detectron2 [1] model which can be used in Pytorch [2] framework easily. Pythorch is a free access, open source framework developed by Facebook's AI team. We investigated other solutions how we could teach models in Tensorflow [3] for similar purposes, but because the simplicity of Pytorch we chose that. We also examined less intelligent options with OpenCV [4], but this only uses outlines made by edge detection from Fourier transformation. There could be numerous problems about this which neural networks can solve this is why we chose the path leading to neural networks.

Implementation

In the next section we would like to introduce our implementation. This will provide a deeper insight of the solution.

1.1 Detectron2

Detectron2 [1] is Facebook AI Research's next generation software system that implements state-of-the-art object detection algorithms. It is a ground-up rewrite of the previous version, Detectron, and it originates from maskrcnn-benchmark. The training of the system will be done with pictures and tags and masks for the pictures from a database. These tags provide accurate rectangle containers for objects, and the masks define the arcs of shapes. In the 1.1 figure we can see that the objects have containers and are masked.

1.2 Datasets

For teaching the model further, we used the free Kaggle [5] databases. The databeses available here are pre-tagged in the format that we need. But this alone is not sufficient, from this our

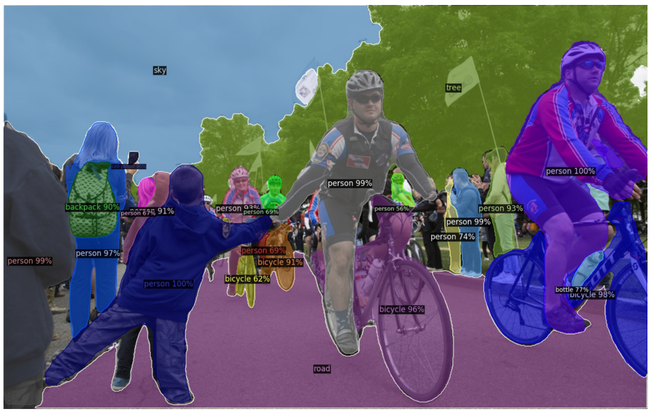


Figure 1.1: Detecting objects on image[1]

model can't feed, so we have to convert this to the appropriate format and we have to generate the describing Json files. We used the free online software Roboflow for this. Roboflow [6] is capable of generating the datasets from the input data. In case of teaching Detectron2 we need Coco type data. So we generated the Coco dataformat from the pictures gathered on Kaggle.

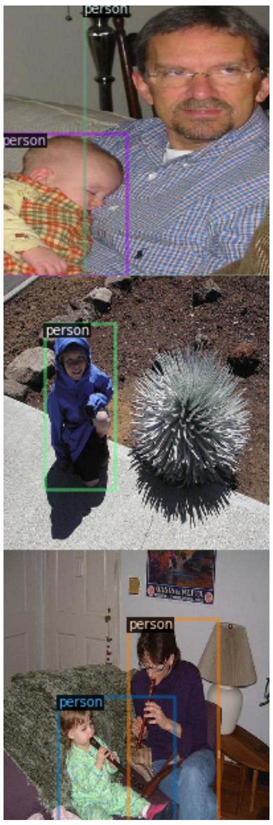


Figure 1.2: Roboflow tagged images

1.3 Distance projection

For measuring the correct distance on the image of the camera, the 3 dimension space must be reconstructed from the 2 dimension image. In order to do this we need to know the focal length and the viewing angle of the camera, which is referred to as "POV”. The letter of focal length is "f” here. In case coordinates X and Y could be determined precisely, we could calculate the distance between the two pedestrian using the Eucledian distance calculus. The scaling is described by these functions:

$$X = u * \frac{Z}{f} \tag{1.1}$$

$$Y = v * \frac{Z}{f} \tag{1.2}$$

The social distance between the two person in 3 dimension space is calculated below

$$\Delta d = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2} \tag{1.3}$$

1.4 Image slicing

The slicing of the videos into frames was done with the open-source Python library, OpenCV [4]. We check the framerate of the input video and we take picture samples according to that. It is important to mention here, that if the framerate is high then the movement vectors are small of the people and the resources are burden unnecessarily, and decreases the number of samples. It would be worth to investigate, how could we correct the sample rate depending on the movement. If the movement vectors are small we decrease the sampling frequency, and if it is high we increase the sampling frequency until we hit a threshold but it can not exceed the video’s framerate. If we would exceed the framerate of the video we would produce redundant image sections carrying no additional information.

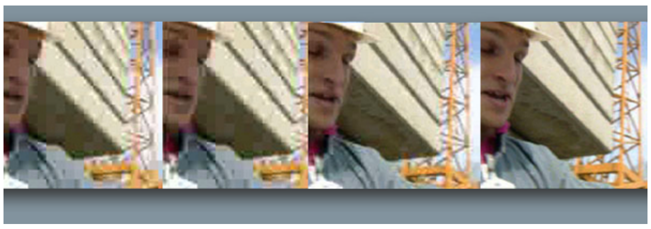


Figure 1.3: Generating image flows from video

Chapter 2

Solution

We can see a frame of the final solution in the 2.1 figure. In the picture we can see three different colours. Blue indicates that the given pedestrian is in safe: there is nobody around them nearer than the given threshold. Yellow indicates a warning: in case of proper mask wearing the pedestrian is not in danger but there is the possibility of getting infected. In this case the pedestrians are connected with a yellow line indicating the possibility of danger. Red indicates pedestrians in danger zone. In these cases maybe proper mask wearing is not enough to avoid getting infected. It is important to mention, that there could be a wrong estimation of distances because of the 3 dimensional aspects due we do not know the exact specifications of the camera.

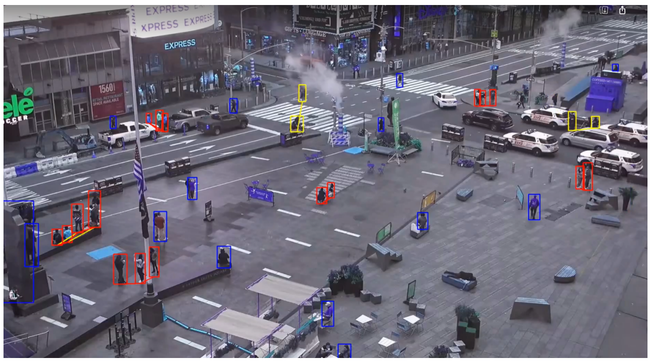
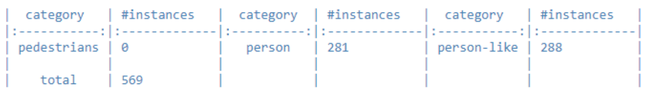


Figure 2.1: Estameted distances

2.1 Train

The output of the teaching is shown on figure 2.2. The database gathered from Kaggle consisted of people and statues. The statues were tagged “person like” and real people were tagged as “person”. In the teaching set there were 281 people and 288 statues. The teaching took about one and a half hour on the Google Colab platform, where an NVIDIA Tesla K80 video card was used. We used the Google Colab platform because in our personal experience it’s the best for running Python Notebook and the system gives us a much stronger dedicated graphics card than the ones in our personal computers. The consumer graphics cards are not made for this.

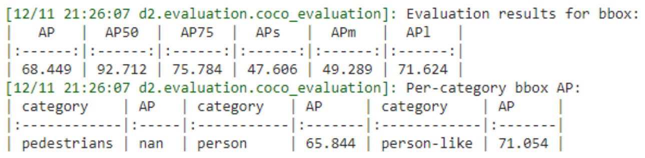


```
|  category   | #instances   |  category  | #instances   |  category   | #instances   |
|:-----------:|:-------------|:----------:|:-------------|:-----------:|:-------------|
| pedestrians | 0            |   person   | 281          | person-like | 288          |
|             |              |            |              |             |              |
|    total    | 569          |            |              |             |              |
```

Figure 2.2: Train results

2.2 Validation

The validation dataset is also from Kaggle. We used the 70 percent of the collected data for training, 20 percent to validating, and kept 10 percent for testing. In the test datas we can see that the model recognize with a 66% accuracy in the person category while the person-like category was with a 71% accuracy. If we could train the model more and with more data, we could achive an even better result, recognizing pedestrians more accurately. Unfortunately due to lack of time and the limits of Google Colab (storage and memory) we could not achive that. In order to develop we should use more powerful resources such as that Azure offers in their Machine Learning Studio service.



```
[12/11 21:26:07 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 68.449 | 92.712 | 75.784 | 47.606 | 49.289 | 71.624 |
[12/11 21:26:07 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category    | AP   | category   | AP     | category    | AP     |
|:------------|:-----|:-----------|:-------|:------------|:-------|
| pedestrians | nan  | person     | 65.844 | person-like | 71.054 |
```

Figure 2.3: Validation results

Google Colab

Colaboratory, or "Colab" [7] for short, allows users to write and execute Python in their browser, with zero configuration required, free access to GPUs and easy sharing.

Azure Machine Learning Studio

Azure Machine Learning is a separate and modernized service that delivers a complete data science platform. It supports both code-first and low-code experiences. Azure Machine Learning studio [8] is a web portal in Azure Machine Learning that contains low-code and no-code options for project authoring and asset management.

Chapter 3

Conclusion

The tool needs some future developments and enhancements. There was no information such as the angle of view of the camera, or the focal distance for the footage that we used. Because the lack of these informations we could achieve the projection from 3 dimension to 2 dimension. This projection would be necessary to esteem the depths of the pictures and with that the eucledian distance could be calculated accurately, that we introduced before in our description. If we would like to apply the solution in the real world we have to solve the problems mentioned before. Furthermore, it would be worth to investigate other models and not only that relies on the Detectron2 and compare the performances of those. A development possibility is to teach the model about statues: for example in our current solution we can see a statue in the left of the screen, which sometimes is recognized as a human. If we could teach the model about statues then it would be more reliable on public squares. We mentioned before that we plan to recognize families, because the system should not go of because of a family since they are not required to keep distance from each other. But this feature was not implemented due to innacurete recognition.

Bibliography

[1] "Detectron2." https://github.com/facebookresearch/detectron2, 2015.

[2] F. A. R. lab (FAIR), "Pytorch." https://pytorch.org/, 2016.

[3] G. B. Team, "Tensorflow." https://www.tensorflow.org/, 2015.

[4] I. Intel Corporation, Williow Garage, "Opencv." https://opencv.org/, 2000. Felkeresve: 2020-12-05.

[5] "Kaggle."

[6] "Roboflow." https://roboflow.com/.

[7] "Google colab." https://colab.research.google.com/.

[8] "Azure machine learning studio." https://studio.azureml.net/.