Final Project | **EPFL**
CIVIL-459 | School of Civil Engineering
Spring 2019 Final Project

Final Project - An Object-Following Robot

1 Introduction

In this project, you will learn to use the concepts we have seen in the lectures and practiced in the labs on a real-world dataset, start to finish. In most machine learning projects, a data scientist starts by using popular architectures and modifying it for his own task and data. In this project, we will be modifying a human pose estimator to perform object detection. Once a good object detector is obtained, it will be used to make our robot, Loomo, track a specific object moving from one point to another in the shortest time possible.

2 Logistics

Group formation. For this project, you will work in a team of 2 students, by your choice. If you are still searching for teammates, please use the discussion forum on Moodle. A good data science team combines a diverse set of skills, and greatly benefits from inter-disciplinary backgrounds.

Deadlines. The deadlines for the different milestones will be announced in class and on moodle.

Deliverables at a glance.

- **Code.** In Python. For this project, we want you to implement and use the methods we have seen in both classes and lab sessions. External libraries and repos can be used, if properly cited. Make sure to preserve the naming of files and functions already provided. Each milestone is submitted independently.
- **Written Report.** You will write a final PDF report on your findings. It includes also the challenges you faced and have solved. It is recommended to use LaTeX to have a properly formatted report.
- **Competitive Part.** To give you a fair ranking, we will be comparing the time the robot needs to go from one point to another. We will also evaluate other features such as detection accuracy and collision avoidance.

The Dataset. The model that will be used, OpenPifPaf[1], is written for the COCO Dataset[2]. Any other dataset can be used if a proper dataset loader is written.

3 Milestones:

Download the zip file corresponding to each milestone and fill the required missing code. The 'pass' statements in the python files indicate that it should be replaced by your own block of code. Feel free to create new files, classes, and functions to help you in your tasks. Make sure that the main scripts provided in each milestone can be called properly.

Note: Make sure to use the v0.5.2 tag of OpenPifPaf which you can set in the branch dropdown in the upper left. (link[1])

3.1 Milestone 1 - Implementing a COCO Dataset Loader for OpenPifPaf

OpenPifPaf[1] is our new bottom-up method for multi-person 2D human pose estimation that is particularly well suited for urban mobility such as self-driving cars and delivery robots. The underlying concept that makes OpenPifPaf a strong human pose estimator can be used for other tasks such as detection. This can be done by writing a proper dataset loader specific for detection, the main task for the first milestone.

We will implement a dataset loader, similar to CocoKeypoints[2], that takes the full COCO dataset and prepares it to be used for detection by OpenPifPaf. The dataset loader requires the use of 'pycocotools' to read the images and annotations, the combination of both bounding box annotations and class labels to create new ground-truth labels, and the proper preprocessing of the images for OpenPifPaf. The new ground-truth labels are created such that there are the same number of PIFs as there are object class labels, and each PIF represents the center of the bounding box of the object.

Note. We will not be using the same annotations as OpenPifPaf. We will be using annotations that contain all the COCO classes with their bounding boxes.

Note2. Currently in OpenPifPaf, the PIFs represent the 17 keypoints. Even though the new ground-truth labels are not keypoints, add these labels with a 'keypoints' key in the final annotations dictionary. This will be clearer once you understand OpenPifPaf and the dataset it uses.

To setup your environment:

1. Install OpenPifPaf by running:

   ```
   pip3 install 'openpifpaf[train,test]'==0.5.2
   ```

2. Follow this file[3] to download the MSCOCO data. Make sure the data-mscoco folder is located in the same directory as your training script.

3. Create a directory called 'outputs' in the same directory as your training script.

In this milestone, the 'train.py' and 'predict.py' run with the same commands in the official OpenPifPaf repo (**openpifpaf.train** and **openpifpaf.predict**) and changing the 'headnets' argument to '--headnets pif80' indicating 80 COCO class labels.

- **Example command to train:**

```
python train.py --lr=1e-3 --momentum=0.95 --epochs=10 --lr-decay 60 70
 ↪  --batch-size=8 --basenet=resnet50block5 --head-quad=1 --headnets pif80
 ↪  --square-edge=401 --regression-loss=laplace --lambdas 30 2 1
 ↪  --crop-fraction=0.5 --freeze-base=1
```

[1]https://github.com/vita-epfl/openpifpaf/tree/v0.5.2
[2]https://github.com/vita-epfl/openpifpaf/blob/v0.5.2/openpifpaf/datasets.py
[3]https://github.com/vita-epfl/openpifpaf/blob/v0.5.2/docs/datasets.md

- **Example command to predict:**

```
python predict.py --checkpoint <location of trained model> <image to test>
 ↪  --show --pif-fixed-scale=1.0 --instance-threshold=0.0
```

3.2 Milestone 2 - Implementing a Custom Dataset Loader for OpenPifPaf

Since this project requires a robot to follow a specific object, it is important to make sure that your model is a perfect detector for this object rather than having a detector that is good for many objects. One way to do this is to prepare your own dataset which can be combinations of different dataset, images you collected, or etc. In this milestone, you will specify an object that you want your model to focus on, prepare a dataset, and then create a custom dataset loader and predictor.

3.3 Milestone 3 - Improving OpenPifPaf Detections

After preparing a dataset and a good detector, the next step is to improve the model's performance. This is where you will be using your background knowledge and information from class to improve your detections. This involves hyper-parameter tuning, data augmentations for both training and prediction, and etc.

3.4 Milestone 4 - Handling Model Failures (Optional)

Although this is an optional milestone, it is rather an important one since it can differentiate your project from other team's project. In this milestone, you will need to handle the different failures of your network such as misdetections. There are many ways to solve these challenges, and this will give your model a good competitive edge.

References

[1] Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. Pifpaf: Composite fields for human pose estimation. *CVPR, arXiv preprint arXiv:1903.06593*, 2019.

[2] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.