# Report third project “Learning-based data fusion”

1. INTRODUCTION

The aim of this final project is to fuse data from two different sensors, looking at the same scene, in a learned trend. So, the actual learning is not only in recognizing the pattern related to the object itself, but also in learn the correlation between the two different types of images.

Disclaimer: since we used the same dataset, we will directly refer to it without writing anything more.

1. YOLOv3 CUSTOM, the GITHUB REPOSITORY

YOLOv3, launched in 2018, further enhanced the model's performance using a more efficient backbone network, multiple anchors, and spatial pyramid pooling. Is based on the previous release, the YOLOv2 by adding some layers. With respect to it the YOLOv3 is faster and more precise, but slightly bigger.

To set up our work we took the python programs from a project (<https://github.com/jas-nat/yolov3-KAIST/tree/master>) made by a GitHub user named Jason Nataprawira, he is a data-science student graduated at the Ritsumeikan University (Japan).

He worked on the KAIST dataset to perform multispectral pedestrian detection. We decided to clone his repository and try to run his code on our HPC.

Into the repository there were:

* Many configurations file to choose from
* A label converter code
* Many .data files
* Train/test/detect codes for different scenarios, such as RGB/infrared only and multispectral ones
* An ‘utils’ folder with many important structure files
* A ‘weights’ folder with a bash file to download different types of weights
* The ‘models.py’ file which contains some important function definitions about weights and network model

1. OUR WORK
   1. DATA PREPARATION

First of all, we used the files ‘train\_night’, ‘val\_night’, ‘train\_night\_ir’, and ‘val\_night\_ir’ to understand which images and labels were needed for this project. All the images reported in these files contained at least one subject, so there were no background images, differently from the dataset used for the first two projects. This was supposed to increase the efficiency of the training. The images were then joined altogether creating a new dataset divided into four folders reporting infrared and RGB images both for training and validation. The labels were converted using the same python program we developed in the first project (since the GitHub one was not working) and placed them in a parallel repository with the same structure as the images one.

* 1. CODE ADAPTATIONS

To adapt the codes provided by the GitHub folder we had firstly to modify some key parameters and data paths into the main file performing the training procedure. In there we specified the new directories of the .data file, .cfg file, and the weights path. The .data file contains the path to the dataset and to the names of the classes contained in the .names file; the .cfg file contains the description of the actual network architecture. The one we have chosen is one of the many files provided by the author able to accept a multispectral input, both the RGB and the IR images, in order to accomplish the learned data fusion task.

This can be demonstrated looking at the configuration file structure of the network we used. It accepts an input of four channels, three of them coming from the RGB image and the last one from the thermal image. We can talk about learned data fusion because the network learns how to combine information from different sources during the training process to improve its image processing and interpretation skills. In this case, since the channels are concatenated at the start of the neural network, and it has all the information about the two images right from the beginning we can conclude that we are performing an “early-fusion”.

In total, all the programs cited before comprehend more than 2’500 lines of code poorly commented, and worse implemented. In fact, we had terrible problems in compiling the programs, some of them had some strange errors and other were only non-recognized relative paths. We spent more than 80 hours trying to make it work, and once we did the metric were terrible, since there were problems not detected in the test program. Once we solved them, we were able to see the metrics of precision and loss provided by our neural network.

* 1. RESULTS

At the end we were able to train the custom network for a total of 30 epochs, divided in three batches of 10 epochs each, which led to not as good results as we expected but neither too terrible ones; in fact, the mAP@0.5 was around 0.5 and an overall precision of 0.63.

As final step, we ran a last code, named detect\_multi, which aim is to submit images to the trained network and save the resulting images on which was drawn the bounding boxes coming from the NN inference.

1. CONCLUSIONS

In conclusion, after spending more than 120 hours working on this project, we thought there was no more that we could do. Unfortunately, we are not able to develop such a complex system from scratch, in particular thinking about the structure of the network itself. Because of this we tried to replicate a work made by others, but since the KAIST dataset is not so famous there was a very small pool of work to choose from, and we got a bit unlucky.

Apart from this we were able to accomplish the requested task, training a network able to lean features both from the infrared and the RGB images.

Valerio Tiri, Federico Rovighi

06/01/2024