Project overview - Tracking Swimming

Théo VINCENT
supervised by Rémi CARMIGNIANI & Vincent LEPETIT

Juillet 2020

The main idea of the project is to obtain the instant speed of the swimmers during a tournament. The video clips should be taken from a fixed camera with no tilt, pan or zoom, as it is shown in Figure 1. There is no condition on the quality of the video or on the camera's position.

The code overview can be seen in the file *reports/code_overview.pdf*

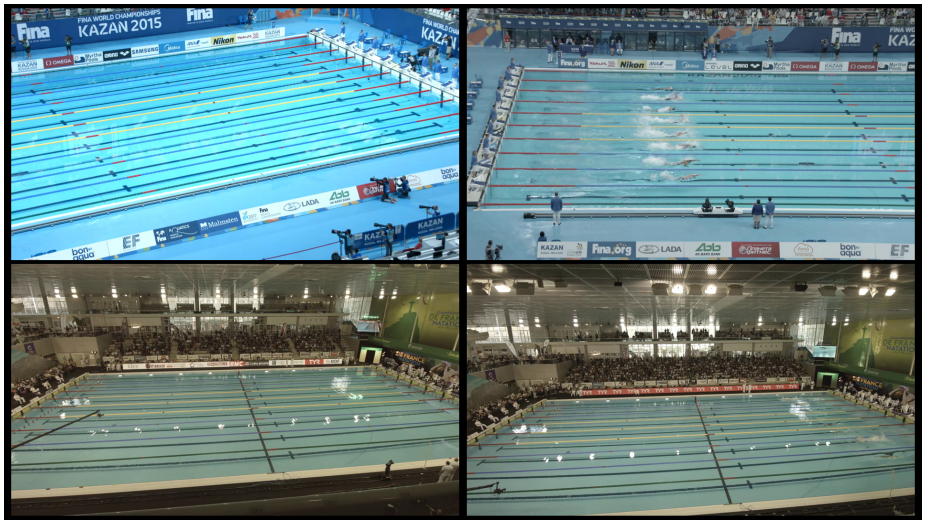

Figure 1: First frame of raw video clips

Video clips are located in the folder *data/1_raw_videos*. The file *list_videos.txt* in this folder maps out all the video clips that are available. The file *interesting_seconds.xlsx* in the same folder gives the seconds where swimmers can be seen for each video. The module *d1_raw_video_summary.py* in *src/* maps out all the video clips that are available in the folder *1_raw_videos* and save this information in the file *list_videos.txt*.

1 Labelling

To train the neural network with a video taken during a tournament, three files are needed:

- A calibration file that will be used to calibrate the video such that we have top-down view images.
- A pointing file that will contain the positions of the swimmers head for every lane.
- A flip file that will contain the information about the direction where the swimmer is going.

The Figure 2 shows examples of such files.

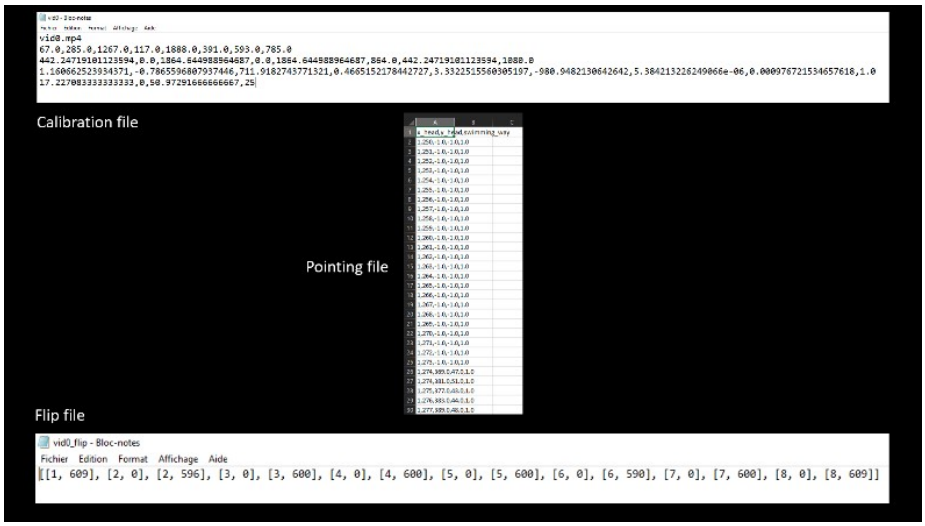

Figure 2: Files that are needed to train the neural network with a video

With those files, every lane of every frame has to be registered on the computer. The lanes have to be with a top-down view. This is done thanks to the calibration file. The result can be seen in Figure 3

Figure 3: A lane with a top-down view

The content of the calibration and pointing files are explained in the files *reports/calibration.pdf* and *reports/pointing.pdf*. These two label files and all the lanes frame can be created with the script *create_data_set.py*. It will register the calibration .txt file in the folder *data/2_intermediate_top_down_lanes/calibration*

the .jpg lanes will be stored in the folder *data/2_intermediate_top_down_lanes/lanes/"video_name"* and the .csv pointing file will be in *data/3_processed_positions*.

2 Pre-processing

The pre-processing stage has been made with the idea of minimising the memory complexity. Thus, the images are only load during the training.

2.1 Before loading

This step is done before the training. A list forms the data set. Each element of this list represents a lane of a frame that has been labelled in the files described previously. An element is defined by the path where the lane is stored (ex : *data/2_intermediate_top_down_lanes/lanes/"video_name"/l"lane_number"_f"frame_number".jpg*), the position of the head on the vertical axis and the horizontal axis, the direction where the swimmer is swimming and the length of the video. This list is created by the module *src/d4_modelling_neural/loading_data/data_generator.py*.

2.2 After loading

The loading step is done during the training thanks to a yielder. Each lane of each frame, that is represented in the list created before the loading, is **flipped** if the swimmer goes toward the left side of the pool. Then, **data augmenting** is performed to reduce the probability of over-fitting. Only the colour is randomly changed. After that, the lane is **standardized**. Here, the idea is only to have values close to zero and a standard deviation close to 1 so 100 is subtracted to the channel and the channels are divided by 50. The lane is then **rescaled** so that the swimmers always have the same size when he/she is passed through the neural network. The imposed scale is 35 pixels per meter. Finally, in order to have a common size, the rescaled lane is **padded**. All these operations are showed in the Figure 4. This yielder is coded in the module *src/d4_modelling_neural/loading_data/data_loader.py*.

At the end of the pre-processing, a yielder is available for the training stage.

3 Processing

The processing stage can be essentially resumed by the training of the neural network, implemented with TensorFlow. When a lane is loaded, it is passed through a sampler that returns a list of smaller images with their labels. These smaller images are taken randomly on the lane. The percentage of small images that contains a head can be chosen. A special variable called *close_to_head* enables to sample small images near the head instead of in the entire lane. This sampler can be find in the module *src/d4_modelling_neural/sample_lane/sample_lane.py*. The Figure 5 shows what this sampler is doing.

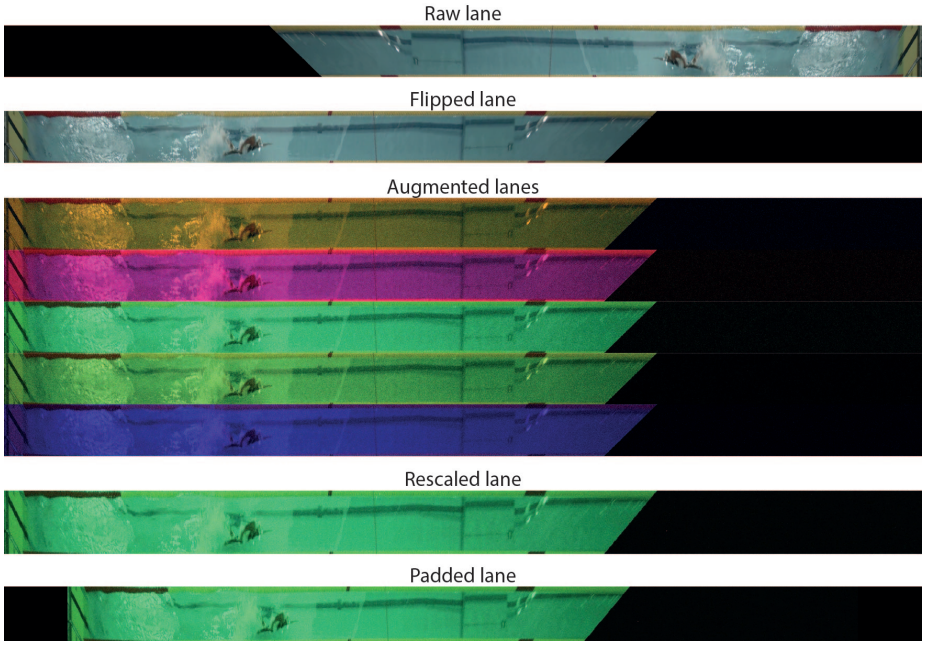

Figure 4: Time line of every operation applied to the input lane after loading

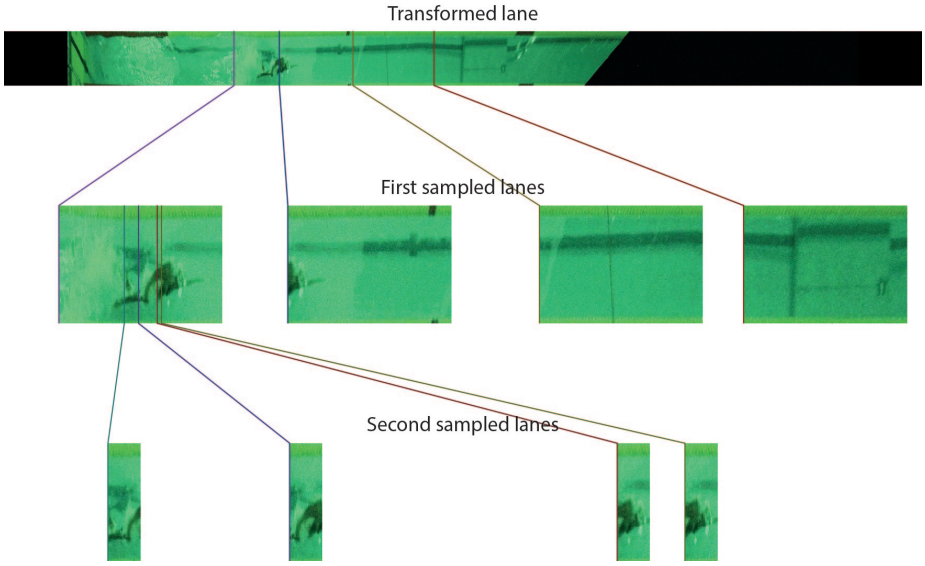

Figure 5: Small images sampled from a lane, the second sampled lanes are computed with the variable *close_to_head* = True

Two different neural networks have been coded. A simple one called ZoomModel and a deeped one called DeepZoomModel. They both return a list when a small image is passed through them. This list is composed of the probability that the head is in the small image, the probability that the head is not in the small image and the prediction on the column where the head is. The column is the relative index column in the small image.

These models can be seen in the files *src/d4_modelling_neural/zoom_model.py* and *src/d4_modelling_neural/zoom_model_deep.py*.

As you can see two problems are addressed with the same neural network : a classification problem (Is the head in the small image ?) and a regression problem (In which column is the head ?). That is why the loss is composed of two terms : the binary cross entropy for the classification problem and the mean square error for the regression problem. A trade-off is there to grant important to one of the two terms. This trade-off has to be tuned. The loss is computed thanks to the module *src/d4_modelling_neural/loss.py*. To tune the trade off, the script *neural_tracking_tune_trade_off.py* can be used.

During the training, the accuracy for the classification problem and the mean absolute error for the regression problem are computed. Graphs of these metrics according to the epochs are registered in the folder *reports/figures_results/zoom_model/"model_type"*. The module *src/d4_modelling_neural/metrics.py* manage those tasks. The Figure 6 shows what it looks like.

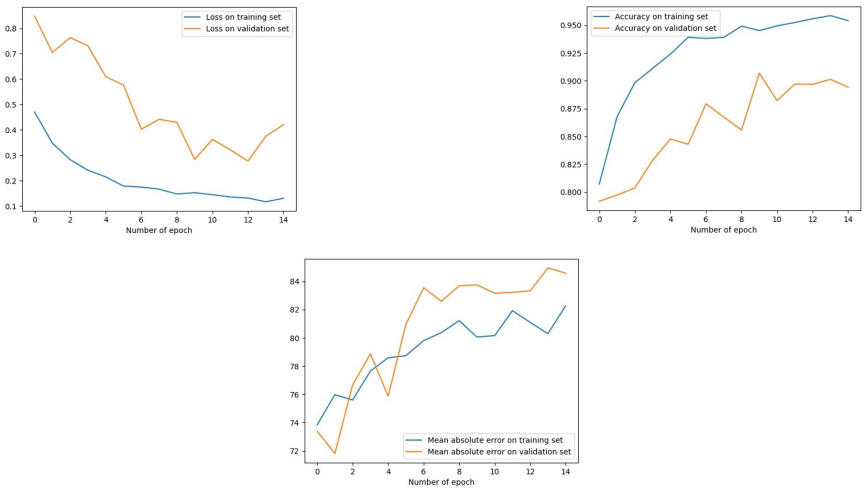

Figure 6: The metrics during a training, the mean absolute error is random since the training was made with $trade\_off = 0$

At the end of this stage, we have a model that can predict the zone and the column of the head.

4 Post-processing

The idea is to use two neural networks. One that has been trained on small images that have 150 pixels on the horizontal axis. Another one that has been trained on small images that have 30 pixels on the horizontal axis. Thus, we can first get a rough prediction to reduce the dimension of the line and then, we can apply the second neural network on a tighter image. The module *src/d5_model_evaluation_magnifier.py* applies this idea.

A video with the predictions and a graphs that plots the time according to the distance can be created. The script *observe_model.py* creates reports and saves the video in the folder *data/5_model_output/videos* (Figure 7) and the graph in the folder *reports_results* (Figure 8).

Figure 7: A screen shot of the visualisation

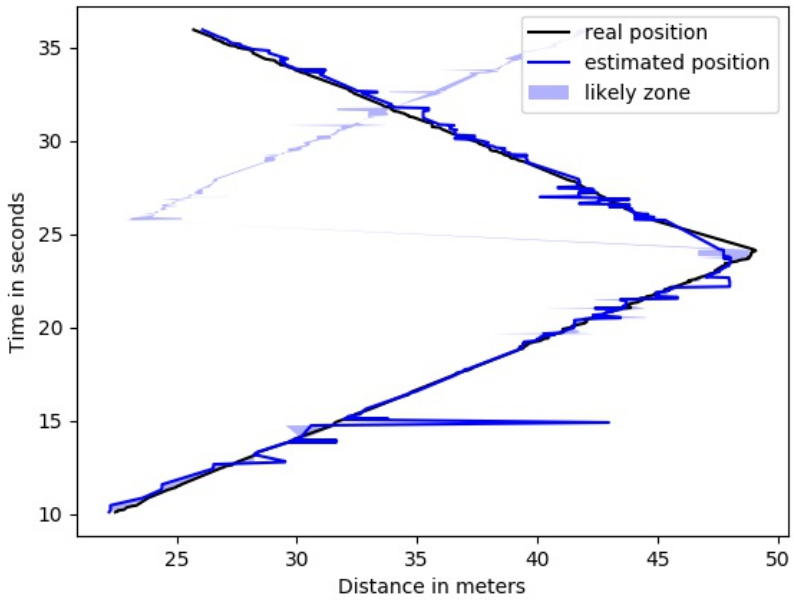

Figure 8: A graphic that show the time according to the distance