Topic 2 - Thermal sensor based human detection model

BEZIAUD Jordan *(214366IV)*,
LEBARON Maëlie *(214322IV)*,
ROSSIGNOL Vincent *(214307IV)*
Department of Computer Systems
Tallinn University of Technology
Tallinn, Estonia

I. INTRODUCTION

In this paper, we will describe the development of a sequential model written in Python using Keras. This model will be used in embedded systems as it will fit into 1-core ARM MCU (in our case a STM32F4 card for development purposes) which add several constraints :

- the size of the model (for accounting the storage limitations on the board)
- the band-with of the model (because of the CPU power which is relatively limited)

The goal of this project is to detect a human being in front of a thermal sensor according to several factors such as the temperature and the shape of the image. The model will be trained using previous recorded data from two thermal sensor, under the form of 32x32 pictures such as the following one :

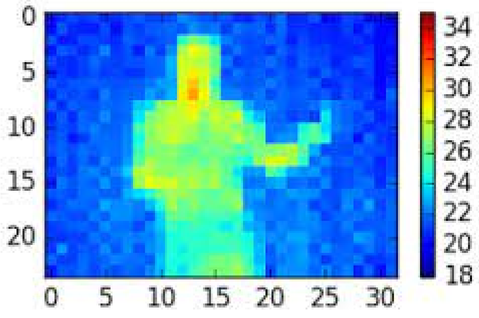

Fig. 1. Sample of data collected from thermal sensors, 32x32 RGB

II. STATE OF THE ART

For years, researchers have been investigating and innovating different applications related to people detection and recognition.

The introduction of machine learning to computer vision problem was a great step forward in the last decade. Furthermore, the recent implementation of optimized neural network for mapping human body pose gives really good results.

Those neural networks were improved using convolutional neural network in order to extract and detect body parts through max/avg pool layer and Conv2D filters. To make the final recognition, the convolutional neural network learnt the hierarchies of the features and made prediction based on that.

Moreover researchers also used semantic segmentation in order to to recognize different forms and body parts. This method gives really good results but highly depends on the quality of the dataset images. Indeed the more pixels there are the more it will be easy to recognize a particular shape.

III. DESCRIPTION OF THE METHODOLOGY

We want to use the several architecture and build several models in this project, as we don't know which one is the most accurate toward our goal. Even though several architectures will be tested out, the constraints regarding the memory size and the computational needs will be kept in mind.
The specific context of the embedded systems require to follow some baseline in regards to the architecture we are gonna explore.

Here are the steps used in our methodology to solve this project :

1) **Data preparation** : figure out how to load the data in data structure that are the most suited to manipulate huge dataset. Then, decide for a way to label it so that we can apply supervised learning and train the coming models.
2) **Data preprocessing** : normalize the data so every values won't create biases when training the model
3) **Model building** : trying different architecture with the same hyper-parameters and take the one that gives the best metrics results (low cost, high accuracy)
4) **Model optimization** : tune the hyper-parameter to get the best learning phase

5) **Model saving and testing** : save the model to different format and load them to the board or in a simulated environment to test the accuracy of the model in real conditions

IV. Implementation

The implementation was done using **python 3.8.10** with *keras* built-in on top of the Tensorflow 2.0 library to train the model and tune the hyper-parameters (with the keras-tuner package).

a) : The first thing we had to do was to pre-process the data. As it was in the form of a .json file, we created a script to transform the file into exploitable data.
We also needed to label the data, so we created a script in Python to help us do it : using the value in the .json, it would plot the data with a thermal map and allow us to enter 0 or 1 depending on if we could identify a human being. It would then create another .json file with a label field added at the end of each line of data.

Aside of the data loading, and normalizing every temperature by the min and max of every feature, the dataset needed to be shuffled in order to avoid some biases especially considering the fact that the input data is a sequence of frame e.g. a video !
Talking about the sources of the data, a problem made us understand the importance of data preparation : when we augmented the size of the dataset by labelling more thermal frames, we forgot to shuffle the dataset. Therefore, when adding to the initial dataset (containing only frames from one sensor) frames from the other sensor, we trained the model once more and got an accuracy of 0.16 without touching the model's settings. It was because the split between the train dataset was made mainly of images from sensor 1 and the test from the other one. Thus, after shuffling the entire dataset and training the model again, the accuracy came back to 0.97 !

b) : After the preprocessing part came the phase of building the actual model.
The data was taking the form of pictures so we thought that using CNN was the right approach, but we decided to start with a basic sequential neural network to have a base to work on, which was our first architecture.

Then we tried to add some layers on the top of the first architecture, such as adding 2 Convolutional layers on top of the classical input layer. An architecture that we called "hybrid".

Afterward, the idea of the CNN came back and we decided to implement it while keeping a small architecture because one of the main constraints of this project is to run the model on embedded systems that have low storage capabilities.

After testing these 3 architectures with some dataset, we figured that the sequential one e.g. the first implemented had a nice accuracy in both training and testing phase. Therefore, we decided to keep this as the model's main architecture.
But in the meantime, we realized that we did not have enough data so we labelled a bit more and we also applied some modification on the data to augment the size of the dataset, such as mirroring the different images.

c) : Then, the last phase was to optimize the model by tuning the hyper-parameters as much as possible to have the best accuracy on our testing set. Using a module built on top of keras, and by trying our own values, we managed to come up with some decent configuration that gives satisfying accuracy on both the train and testing phase.
NB : While we were trying to optimize our model, we tried out to fight the over-fitting to prevent some biases from happening by adding regularization with the L2 and dropout techniques. The problem was that we set those parameters to strong values resulting in a blocked accuracy near 0.5, our model was randomly making predictions about the data ! We then decided to apply regularization but with lower values to prevent huge biases.

d) : In the final model, we had the following parameters for our main neural network that was gonna end up on the STM32 board:

```
#test_size = 300 and training_size=1200

DEFAULT_PARAMETERS_EXTENDED = {
    "INIT_LEARNING_RATE": 0.012,
    "OPTIMIZER": "adam",
    "EPOCHS": 200,
    "BATCH_SIZE": 100,
    "ACTIVATION_FUNCTION": "relu",
    "N_HIDDEN_LAYERS_NEURONS": 80,
    "N_HIDDEN_LAYERS": 2,
    "DROPOUT_PERCENTAGE": 0.1,
    "L2_REGULARIZATION": 1e-4,
}
```

The dropout function is here so that the model don't over fit the data collected.

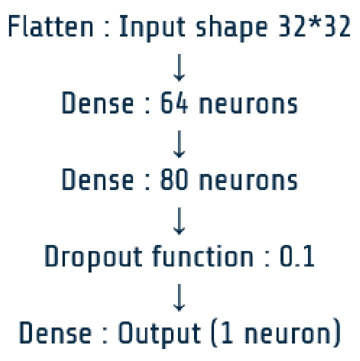

Fig. 2. The sequential model at the end of the project

V. Analysis and Results

To get started, we only had 200 labeled data of the first dataset to make our model run. The accuracy and the loss were about 0.68 and 0.2. Then we suspected that the dataset size wasn't large enough to train the model correctly, therefore we labeled a lot more data to compensate for the previous issue. As a consequence we had an accuracy of almost 0.75.
In order to have more data, we decided to perform data augmentation by duplicate the dataset and invert them to have new images. In that way, **we multiplied by two our dataset length**. At the end of this whole process we managed to have an accuracy of 0.88. After that we were only tuning the hyper parameters such as the batch size in order to find the more suitable one. Eventually we managed to have an accuracy of $92 - 93\%$.

Here is the result of the final configuration and architecture with 200 epochs, $TRAIN\_SIZE = 1200$ and $TEST\_SIZE = 300$:

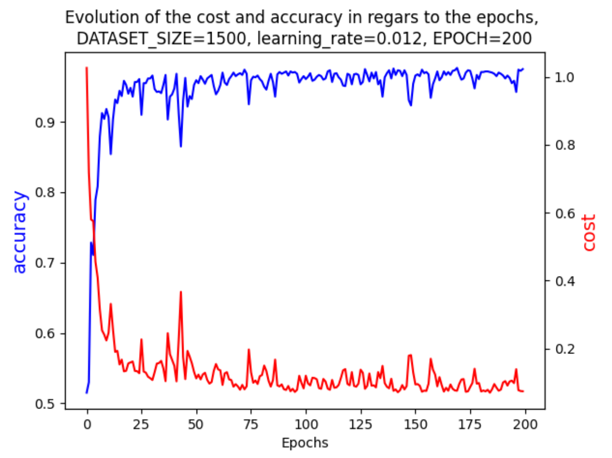

Fig. 3. Evolution of the NN's metrics (cost, accuracy) with $EPOCH = 200$

Furthermore, we get the following accuracy for the validation set :

```
=========== TESTING PHASE ===========
10/10 - 0s - loss: 0.2320 - accuracy:
 ↪  0.943 - 145ms/epoch - 14ms/step
```

Lastly, the model have been converted in tflite model using the built-in converter in the tensorflow library and uploaded it to the board, and as expected the size is decent for an embedded system and can fully fit inside the storage :

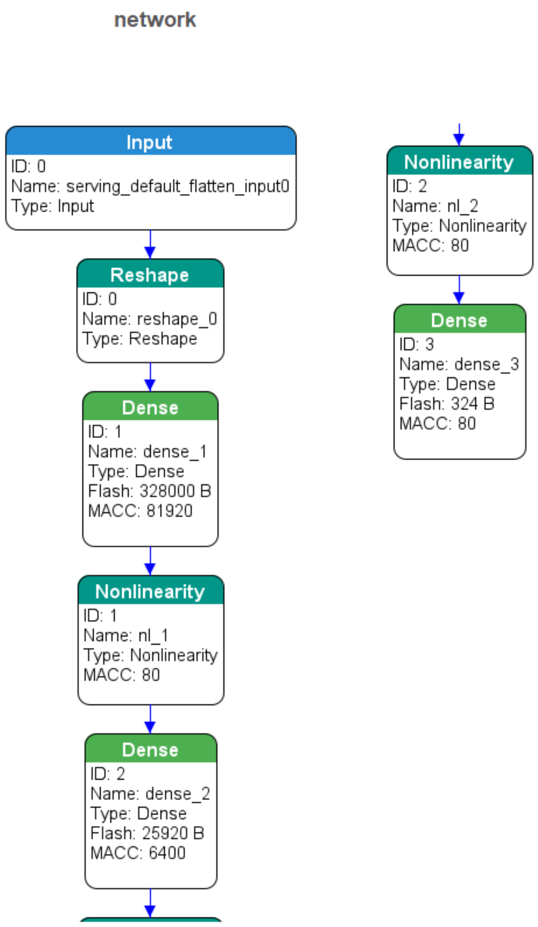

Fig. 4. Representation of the model in STM32

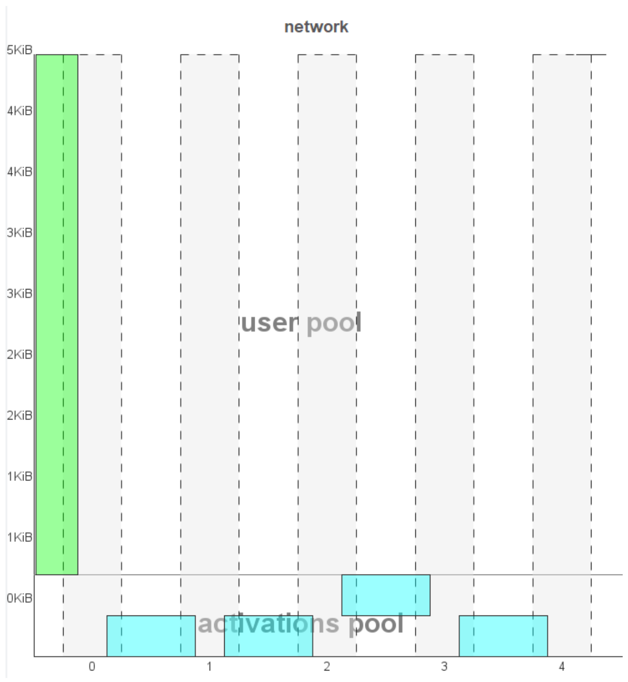

Fig. 5. Memory usage of the model

VI. Conclusions and Future Research

A. Conclusion

In this paper we have proposed our solution to detect a human being using en thermal based sensor.

The model we provided gives a fairly good accuracy, but could still be improved. The dataset size could also help to improve the model. It could also be a great idea to gather data from several other sensor, in order for the model to not over fit the data coming from the two used sensors.

B. Limitations

The main limitation result in the fact that the label was labelled by hand, and only by one human being. In order to have a better accuracy on data coming from different sensor, it could be a good idea to let the data be labelled by different people, or to implement unsupervised model.

Another limitation reside in the model we use, which is Sequential. Using convolutionnal neural network could help to have a higher accuracy as it might detect the right patterns specific to the human shape we're looking for. RNN (Recurrent Neural Network) might also work pretty well as the input is a sequence of frame that follow through time, so the idea of taking in consideration the previous input to predict an output might give interesting results.