# Large Scale Computing: Individual Project

# Human Activity Classification

Valérie BLANCH

2614867B

## 1. INTRODUCTION

The dataset analysed in this project contains data recorded from a smartphone that has been attached to the waist of 30 volunteers aged 19-48. The authors of this experiment are Davide Anguita, et al. from the University of Genova.

With a tri-axial accelerometer and a tri-axial gyroscope embedded in it, different movements have been recorded in 3 dimensions during daily activities.

Each activity has been encoded as follow:

1. WALKING
2. WALKING_UPSTAIRS
3. WALKING_DOWNSTAIRS
4. SITTING
5. STANDING
6. LAYING

The dataset is composed of the following elements:

- One folder for the training set (70% of the observations – 21 individuals) and one for the test set (30% - 9 individuals). Each of them contains:
- tables of time series: 3 tables (for the 3 directions – coded *x*, *y*, and *z*) for total acceleration (accelerometer data), 3 tables for body acceleration (total acceleration minus gravity), and 3 tables for the gyroscope data,
- and the corresponding labels kept in a separate file.

The objective of this project is to predict categories of daily activities from the signals, using neural networks with TensorFlow and the Keras API.

## 2. EXPLORATORY ANALYSIS

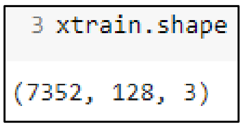
- *Pre-processing:*

For the first parts of this analysis, only the data concerning body acceleration will be studied.

Four NumPy arrays have been created from the .txt files available, two for the training and test set containing the time series, and two for the target variable.

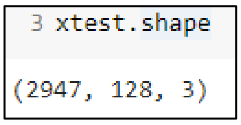
Here are their dimensions :

- *xtrain* is the training set: it is an array containing 7352 3D time series of 128 data points.



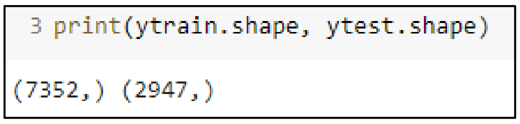
```
xtrain.shape

(7352, 128, 3)
```

- *xtest* contains 2947 3D time series of 128 data points.



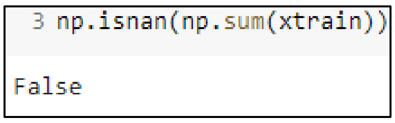
```
xtest.shape

(2947, 128, 3)
```

- *ytrain* and *ytest* are 1D arrays for the target variable.

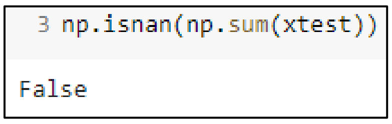


```
print(ytrain.shape, ytest.shape)

(7352,) (2947,)
```

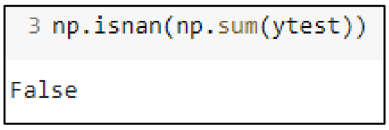
The datasets have been tested for NAs, there were none.



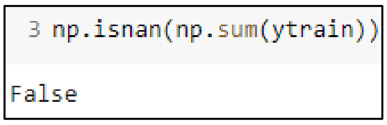
```
np.isnan(np.sum(xtrain))

False
```



```
np.isnan(np.sum(xtest))

False
```



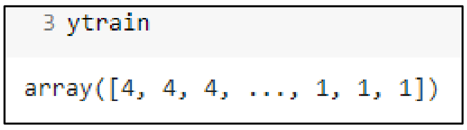
```
np.isnan(np.sum(ytest))

False
```



```
np.isnan(np.sum(ytrain))

False
```

- *Target variable transformation:*

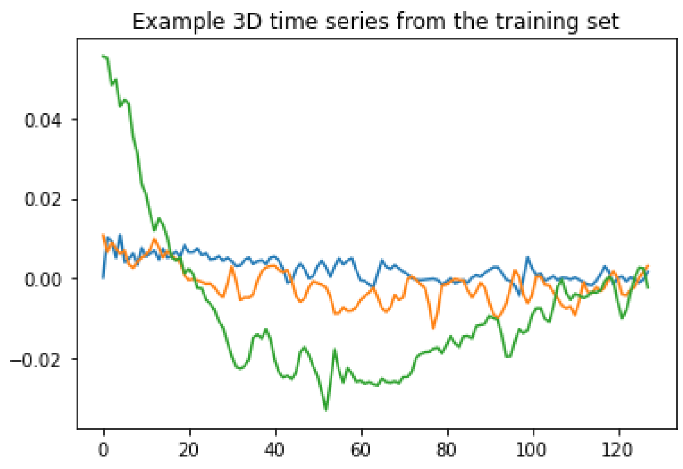
The values of the target variable, initially floating-point numbers, have been converted to integers, to allow for classification. The array and its values look like this:



```
ytrain

array([4, 4, 4, ..., 1, 1, 1])
```

- *Time series visualisation:*

A line plot of the first time series of the training set has been drawn as an example. The x axis represents time, while the recorded values are represented on the y-axis; they have been scaled by the authors of the dataset. Each line corresponds to one activity in one particular dimension, *x*, *y,* or *z*.



## 3. MODELLING

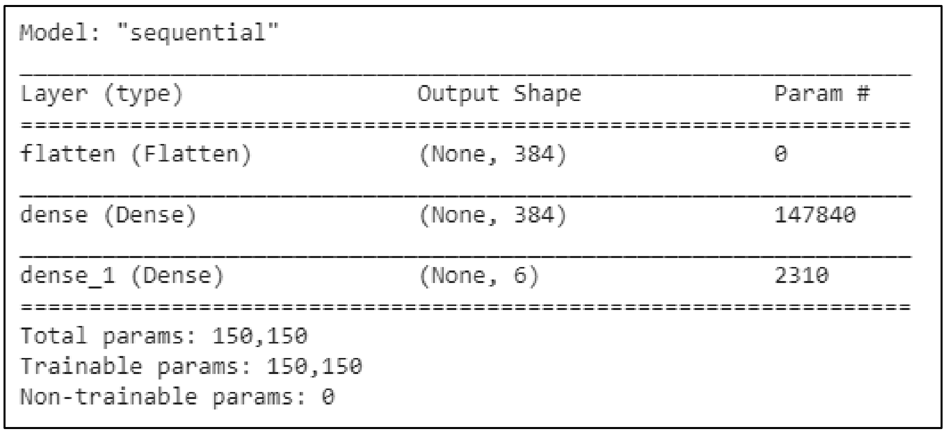
- *Part 1: Body Acceleration – 1-layer Neural Network:*

The objective of the first part of this project is to build a 1-layer NN classifier to predict categories, from body acceleration only.

It has been built using a flatten layer for the input tensor, to obtain a 1D tensor of 384 points, then a dense (=fully connected) layer of 384 neurons with a rectified linear activation function, and finally a dense output layer of 6 neurons with a softmax activation function, corresponding to the six categories to predict.

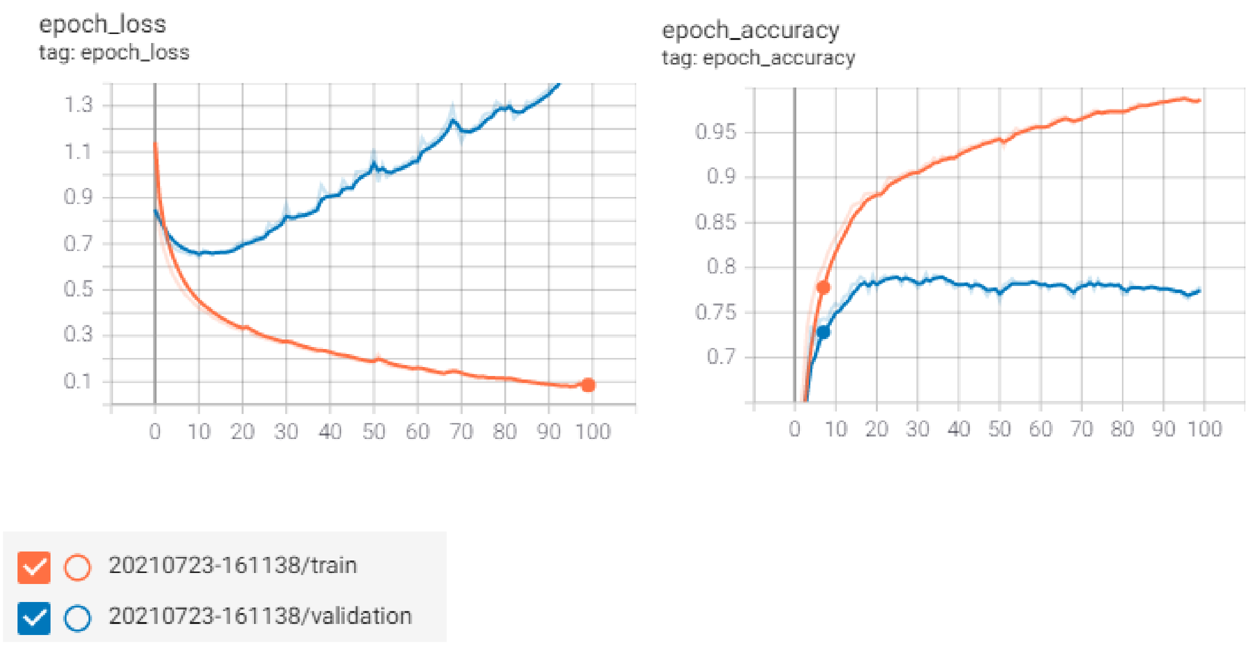
The optimizer used is the Adam optimizer, a gradient descent algorithm with optimal computational power. The learning rate has been set to 0.001, which is the default value and seemed appropriate during the testing process.The loss function has been set to sparse categorical cross-entropy, and accuracy will be the main metric used by the NN for the training process.

Here is the summary of the structure of the network:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 384)               0
_________________________________________________________________
dense (Dense)                (None, 384)               147840
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 2310
=================================================================
Total params: 150,150
Trainable params: 150,150
Non-trainable params: 0
```

To evaluate the NN, the accuracy and the loss on the training and test sets have been computed at each iteration (100), then plotted on TensorBoard:



The model seems to reach an optimum around 15 epochs, then starts to overfit (the loss increases while the accuracy stagnates). After searching the history log, the number of iterations kept for this network is 17. For 17 iterations, the model gives the following predictive performances:

```
loss: 0.3536 - accuracy: 0.8723 - val_loss: 0.6626 - val_accuracy: 0.7900
```

A loss of 0.3536 and an accuracy of 87.23% for the training set; a loss of 0.6626 and an accuracy of 79% for the test set. Those performances are rather modest: they can be explained by the fact that only the body acceleration has been studied by the NN. Furthermore, the network is very simple and contains only one hidden layer.

- *Part 2 – 1D Convolutional Neural Network:*

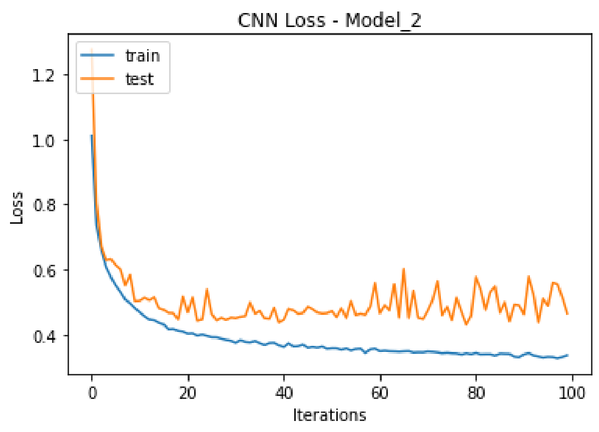
To increase the performances obtained above, a CNN has been built as follows:

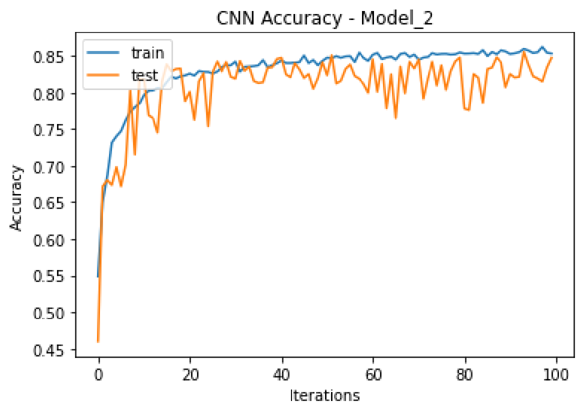
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 125, 32)           416
_________________________________________________________________
batch_normalization (BatchNo (None, 125, 32)           128
_________________________________________________________________
dense_4 (Dense)              (None, 125, 100)          3300
_________________________________________________________________
global_average_pooling1d (Gl (None, 100)               0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 606
=================================================================
Total params: 4,450
Trainable params: 4,386
Non-trainable params: 64
```

- A 1D convolutional layer as the input tensor, with 32 filters and a kernel size of 4, and a ReLu activation function,
- A batch normalisation layer,
- A dense layer of 100 neurons with a ReLu activation function,
- A global average pooling layer,
- A dense layer for the output layer, with 6 neurons and a softmax activation function.

The Adam optimizer with a learning rate of 0.001 has been used again, along with a sparse categorical cross-entropy loss function, and accuracy as the determining metric.

The CNN has been trained over 100 iterations, which gave the following accuracies and losses, plotted here:





The CNN seems to be optimal around 20 iterations, then starts to overfit. The history log gave the following optimum, for 19 epochs:

```
loss: 0.4119 - accuracy: 0.8221 - val_loss: 0.4469 - val_accuracy: 0.8327
```

The accuracy on the validation set is higher than for the previous NN: 0.8327 against 0.79. The loss is lower as well. Therefore, it is a “better” model, from a predictive standpoint.

However, it is possible to modify the CNN to increase its performances even further.

- *Part 2 – CNN model – Upgraded:*

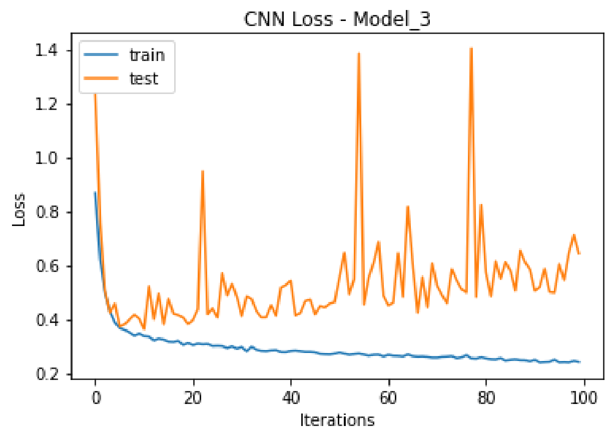
The previous CNN has been modified as follows:

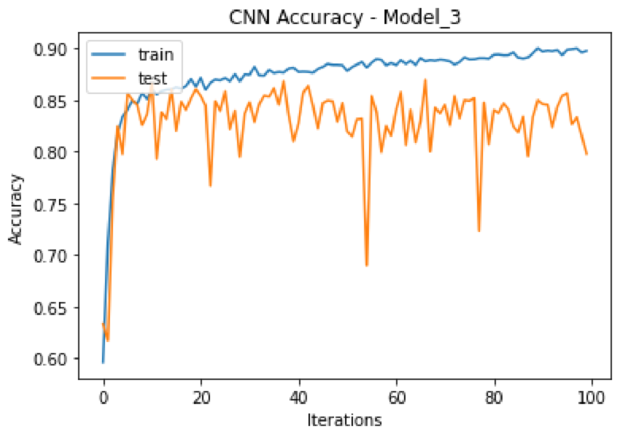
- A 1D convolutional layer has been added
- For both Conv1D layers, the filters have been set to 96 and the kernel size to 2.
- A dropout layer has been added after the dense 100-neurons layer to lower the overfitting, with rate 0.5.

Here is the structure of the updated CNN:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_1 (Conv1D)            (None, 127, 96)           672       
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 126, 96)           18528     
_________________________________________________________________
batch_normalization_1 (Batch (None, 126, 96)           384       
_________________________________________________________________
dense_6 (Dense)              (None, 126, 100)          9700      
_________________________________________________________________
dropout (Dropout)            (None, 126, 100)          0         
_________________________________________________________________
global_average_pooling1d_1 ( (None, 100)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 6)                 606       
=================================================================
Total params: 29,890
Trainable params: 29,698
Non-trainable params: 192
```

Loss and accuracy have been computed the same way as previously, and the following plots have been drawn from them:





The log history gave the following optimal metrics during the 15th iteration:

```
loss: 0.3235 - accuracy: 0.8603 - val_loss: 0.3807 - val_accuracy: 0.8605
```

The test set accuracy is several points higher than for the previous CNN and the loss is lower as well. Therefore, it this the “best” model so far.

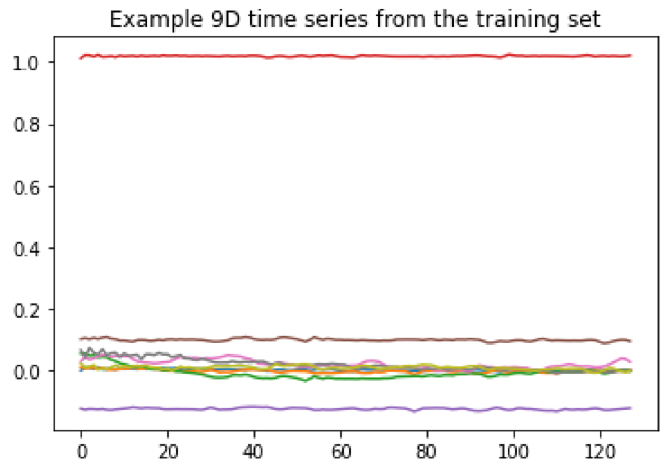
- *Part 3 – CNN model – Full dataset*

In this section, the full dataset has been fed to the CNN. That is, not only the body acceleration time series, but the total acceleration and gyroscope data as well.

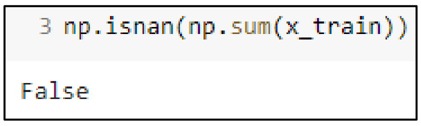
New 3D arrays have been created, with the following names and dimensions:

```
36 print(x_train.shape, x_test.shape)

(7352, 128, 9) (2947, 128, 9)
```

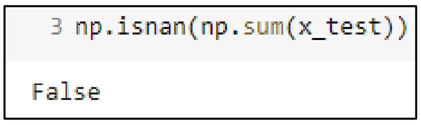
The last dimension of the arrays shows that 9 time series are recorded for each observation. Below is a plot of the first observation of the training set:



The sets have been tested for NAs, there were none.

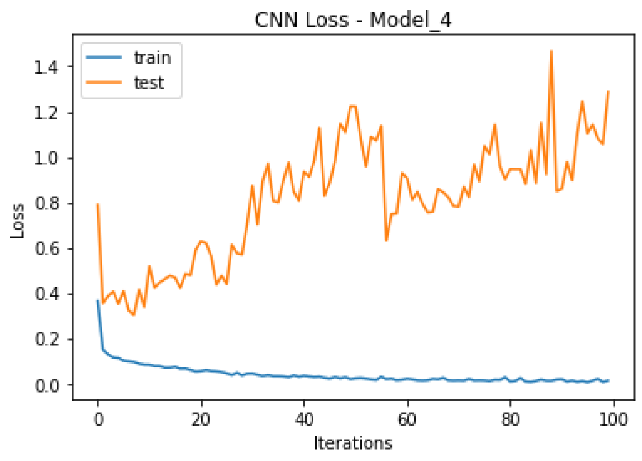


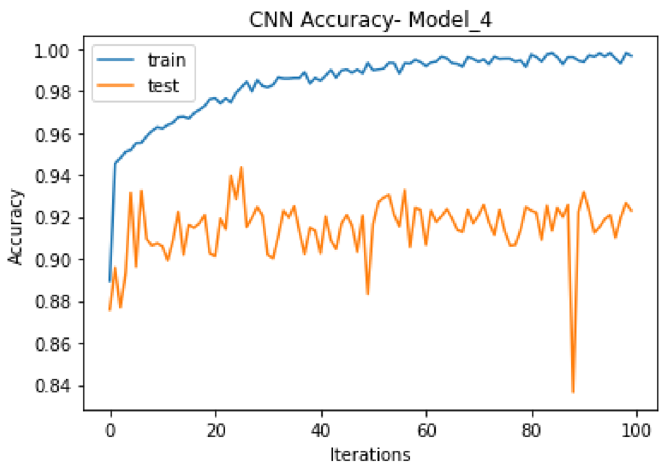
```
np.isnan(np.sum(x_train))

False
```



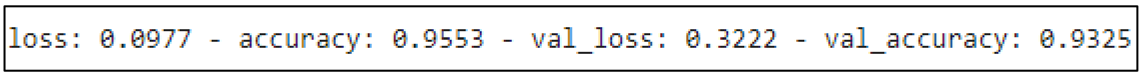
```
np.isnan(np.sum(x_test))

False
```

Feeding the new sets into the CNN, it gave the following loss and accuracy over 100 iterations:





An optimum has been found in the history log, at the 7th iteration:



```
loss: 0.0977 - accuracy: 0.9553 - val_loss: 0.3222 - val_accuracy: 0.9325
```

## 4. DISCUSSION

| | Loss | Accuracy |
|---|---|---|
| 1-Layer NN | 0.6626 | 0.7900 |
| 1D CNN | 0.4469 | 0.8327 |
| 1D CNN – Upgraded | 0.3807 | 0.8605 |
| 1D CNN – Upgraded – full dataset | 0.3222 | 0.9325 |

Given the metrics regrouped in the table above, the “best” model for this dataset is the last one, since it has the lowest loss and the highest accuracy.

This isn’t surprising considering the full data has been used, and the CNN has been upgraded in part 2 to fit this particular dataset.