Cloud Atlas

An LstmEncoder for UHECR AirShowers

G. Becuzzi L. Papalini

July 2022

Table of Contents

1. Introduction
2. Preprocessing
3. Neural Network building

Table of Contents

1 Introduction

2 Preprocessing

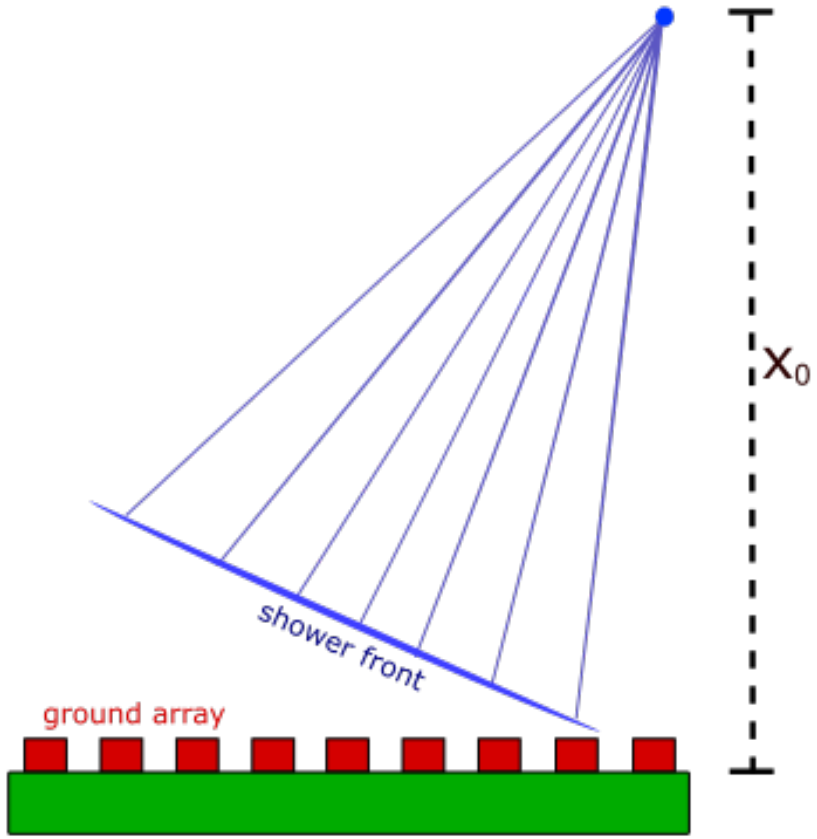
3 Neural Network building

UHECR Airshower

When *Ultra High Energy Cosmic Rays* (UHECR) enters the atmosphere they produce a particle cascade.

Detection: grid of water-Cherenkov ground based detectors.

Prediction: X_0 height at which the shower forms.



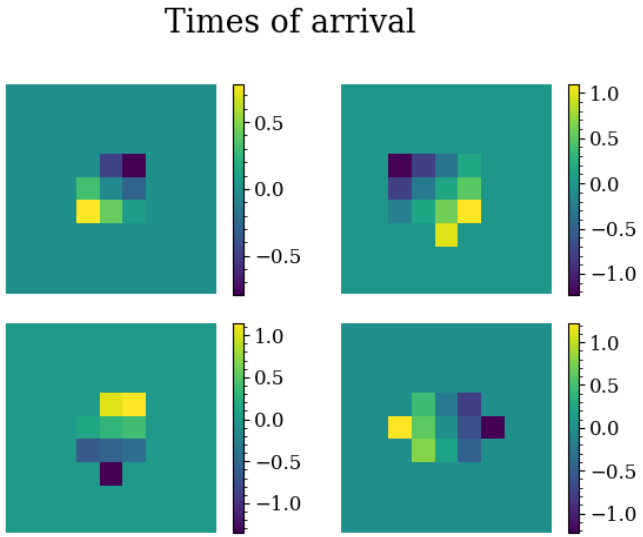
Dataset, first glance

Dataset: 10^5 simulated events:

- 9x9 grid of detectors
- most intense at center
- 80 frames of time series (40 *MHz* sampling rate)
- 1 frame of times of first arrival

Single record shape: $(80 + 1, 81)$

`pd4ml` package splits by default in 70% train 30% test



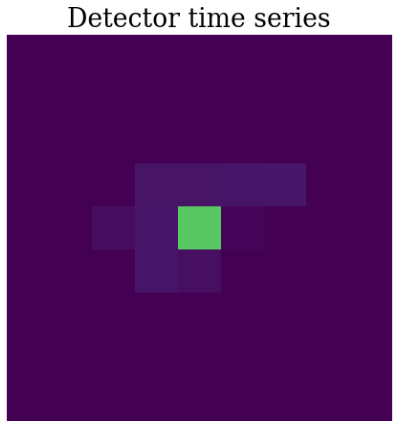


Table of Contents

1 Introduction

2 Preprocessing

3 Neural Network building

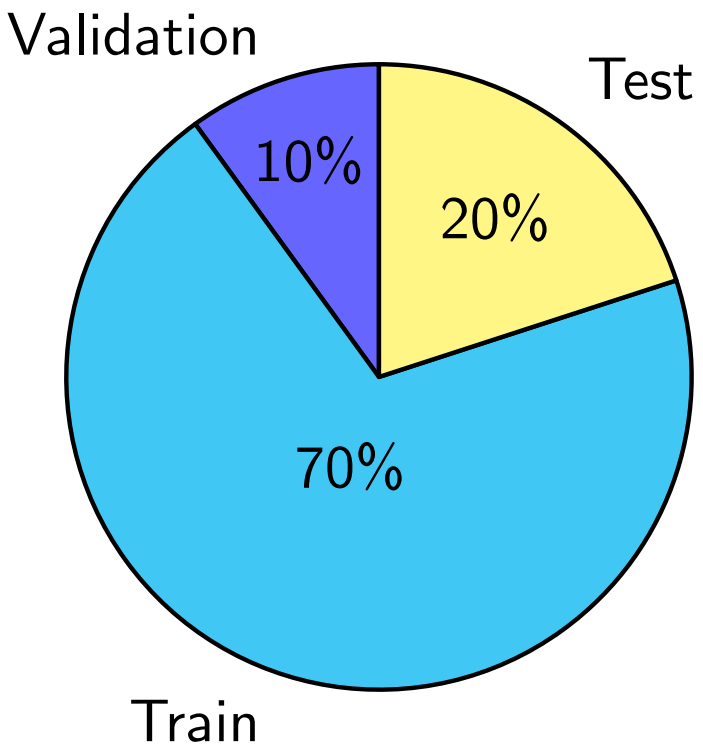
Split the dataset

Dataset was already split in test and train.

We put it all back together, shuffled it and divided with the following percentage:

- 70% *train*
- 20% *test*
- 10% *validation*

For the design of the net it is convenient using `numpy` structured arrays



Split the dataset: `funky_dtype`

```
    # custom numpy dtype
    funky_dtype = np.dtype(
        [
            ("outcome", np.float64),
            ("time_series", np.float32, (80, 81)),
            ("toa", np.float32, (9, 9, 1)),
        ])

```

All data relative to a single event is clustered in a single `numpy` object, transformation is:

$$(80+1, 81) \longrightarrow [("toa", (9,9,1)), ("timeseries", (80,9,9))]$$

Data can be accessed "as a dictionary", depending on what is needed.

DataFeeder class

Ensures an easy way to train the subnets separately Class `DataFeeder` main features:

- shuffles data randomly
- input fields can be specified
- can be extended to more complex training strategies
- returns a generator

Using a generator (`keras.utils.Sequence`)

- inherit multiprocessing features
- has default callbacks
- avoids memory overload

Data Augmentation

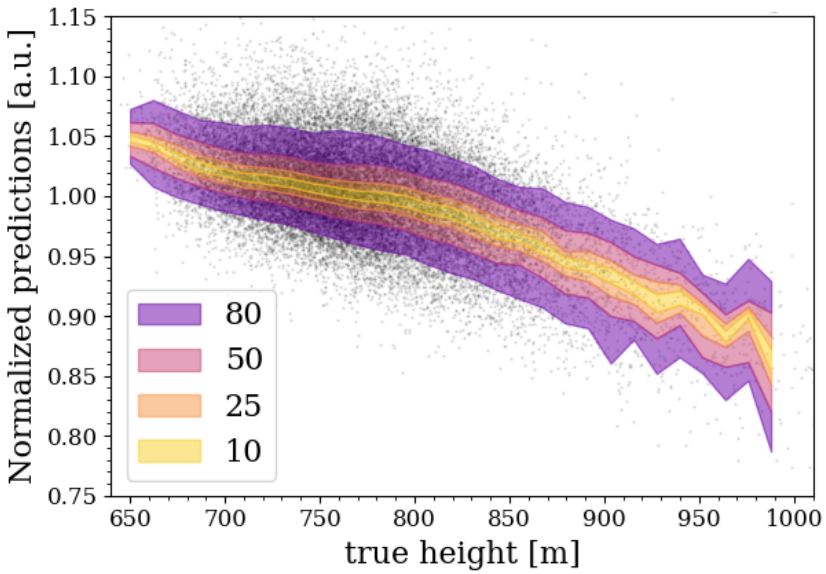
Dataset has a lack of high events ($X > 850m$) so a first network training showed a worse resolution for samples corresponding to this range

Strategy

Increase the number of samples that overcome a certain heigth threshold using the symmetries of the problem

Data is augmented using

- flip up-down
- flip left-right
- diagonal flip
- rotation of 90°



Resolution

The reference article suggests using the resolution:

Resolution

defined as the standard deviation of the distribution given by the difference between the predictions and the actual values of X_{max}

We point out that

$$\sigma^2 = \frac{1}{N}\sum_i (\delta_i - \bar{\delta})^2$$

is a sensible estimator of the average error on predictions only if $\bar{\delta} = 0$, for which the adopted resolution is equal to the *RMSE* of the distribution

$$RMSE^2 = \frac{1}{N}\sum_i (x_i - \hat{x}_i)^2$$

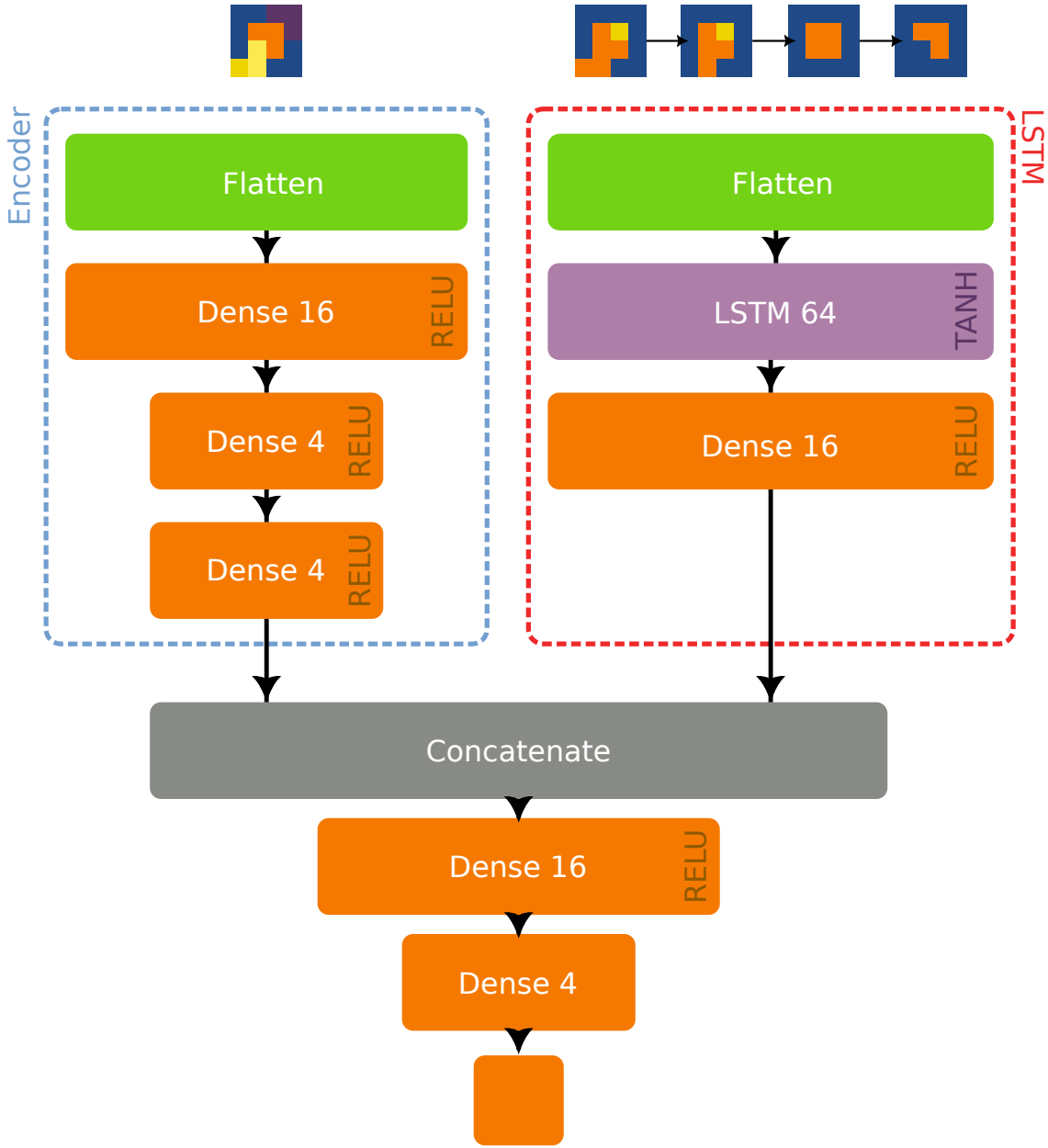
Since (on a typical trained model) $\bar{\delta} \approx 10$m we preferred the RMSE.

Table of Contents

1 Introduction

2 Preprocessing

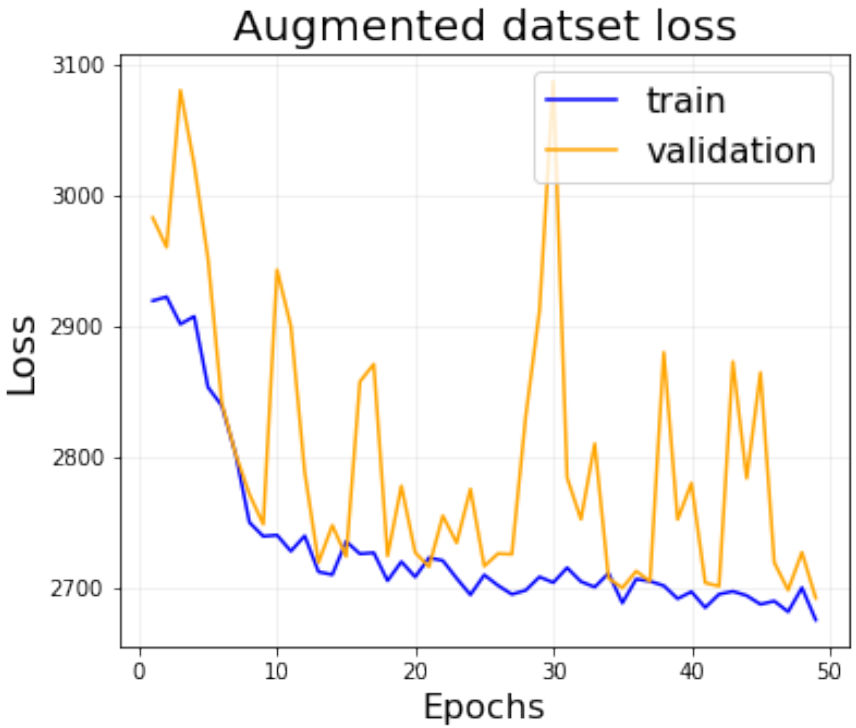
3 Neural Network building

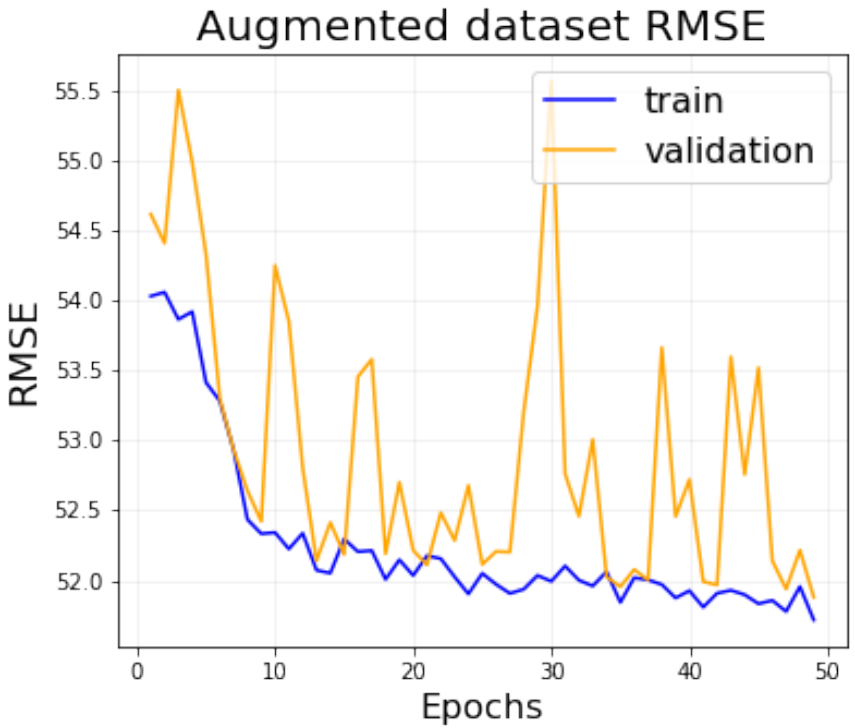

Encoder
Flatten
Dense 16
RELU
Dense 4
RELU
Dense 4
RELU
LSTM
Flatten
LSTM 64
TANH
Dense 16
RELU
Concatenate
Dense 16
RELU
Dense 4

Overview on the network

The assumption that lead to this design is that from the time of arrival matrix it is possible to infer some kind of "homogeneous" shower parameters (incidence angle, spread, etc.) while the time series can be processed by a recurrent network.

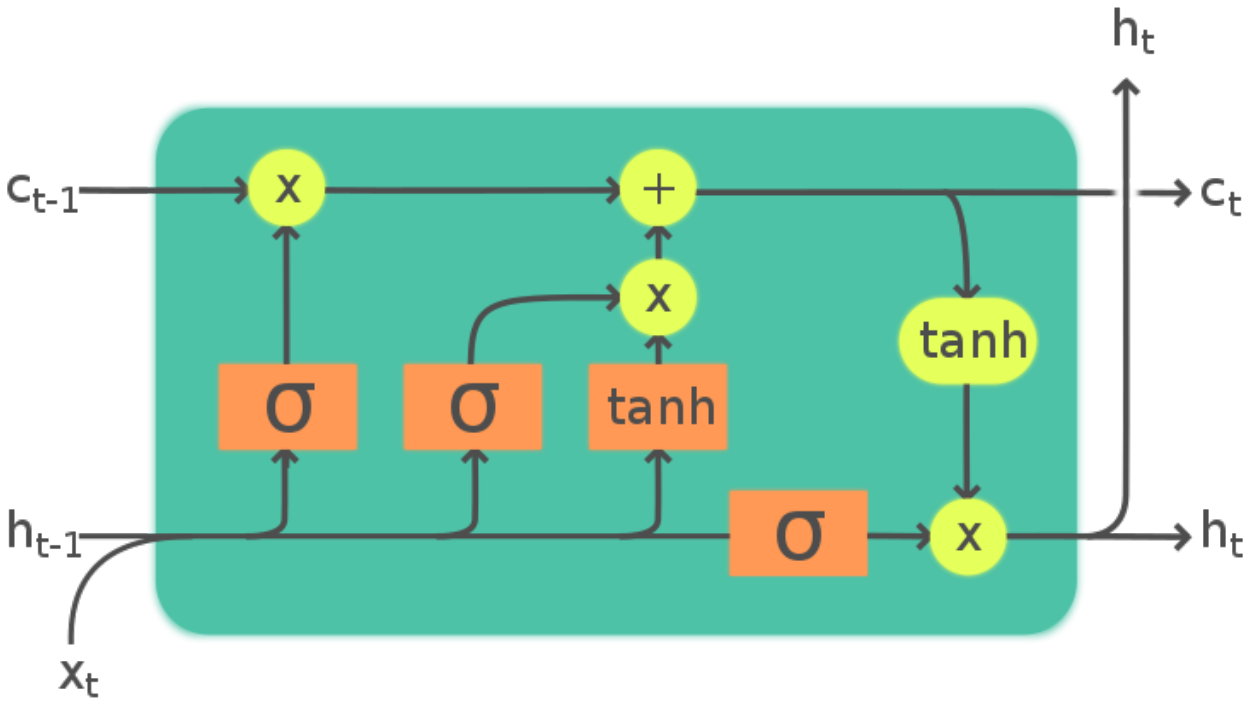
Encoder for the time of arrival



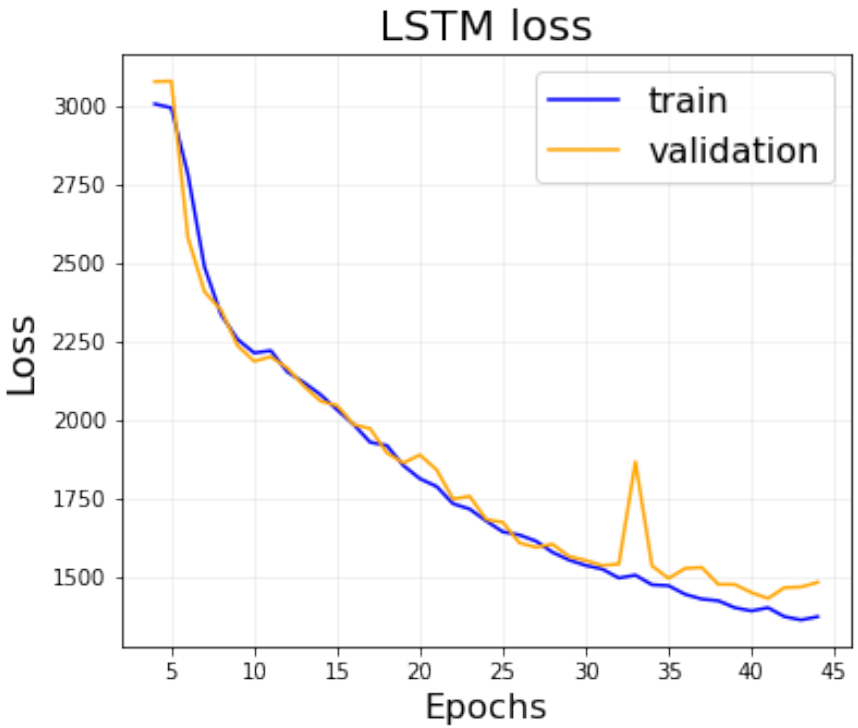


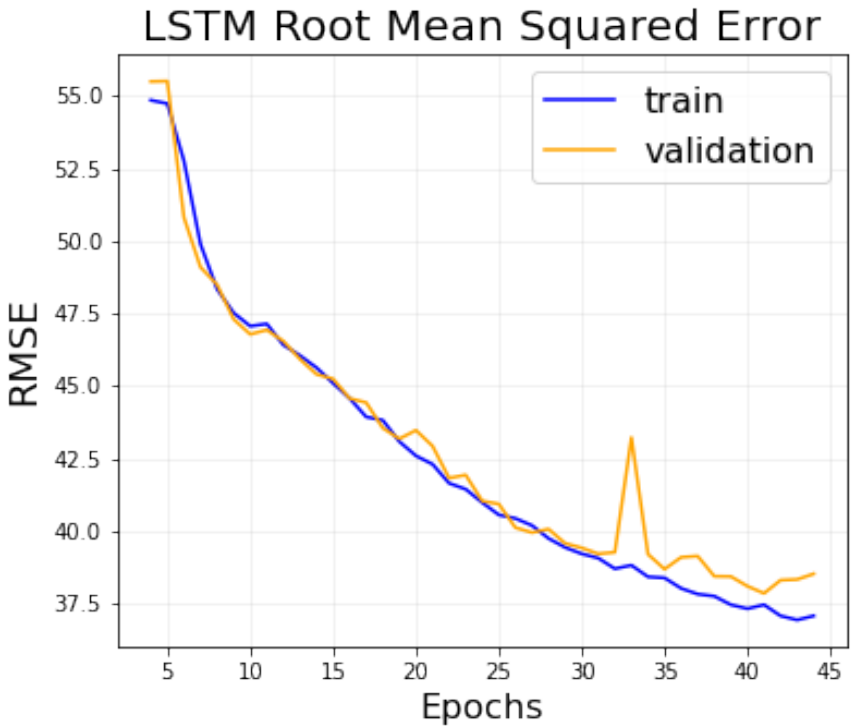
LSTM

LSTM (*Long Short Term Memory*) cell is a variant of a typical recurrent RNN cell. It is able to learn long-term dependencies that brings along in a hidden state.

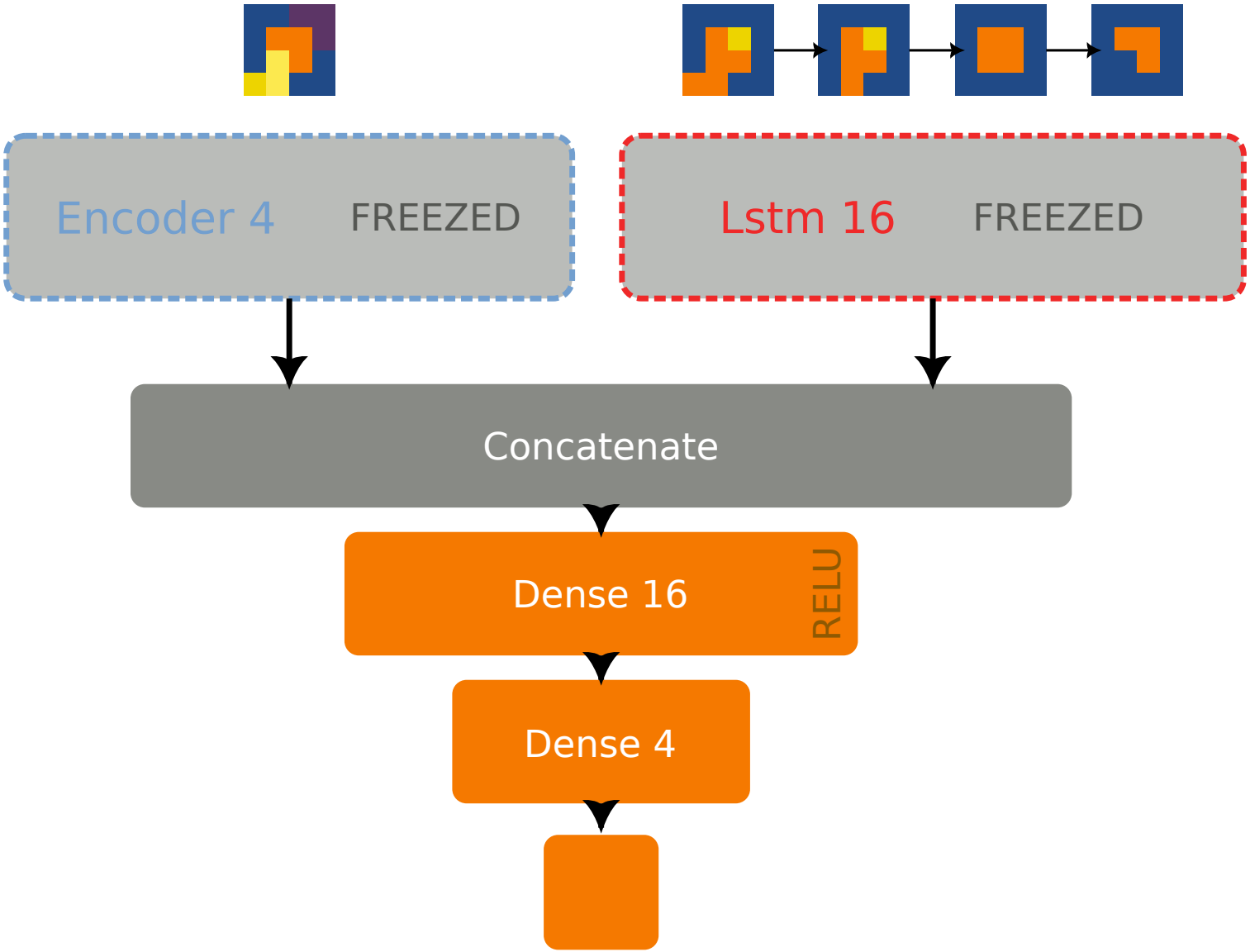


LSTM for the time series



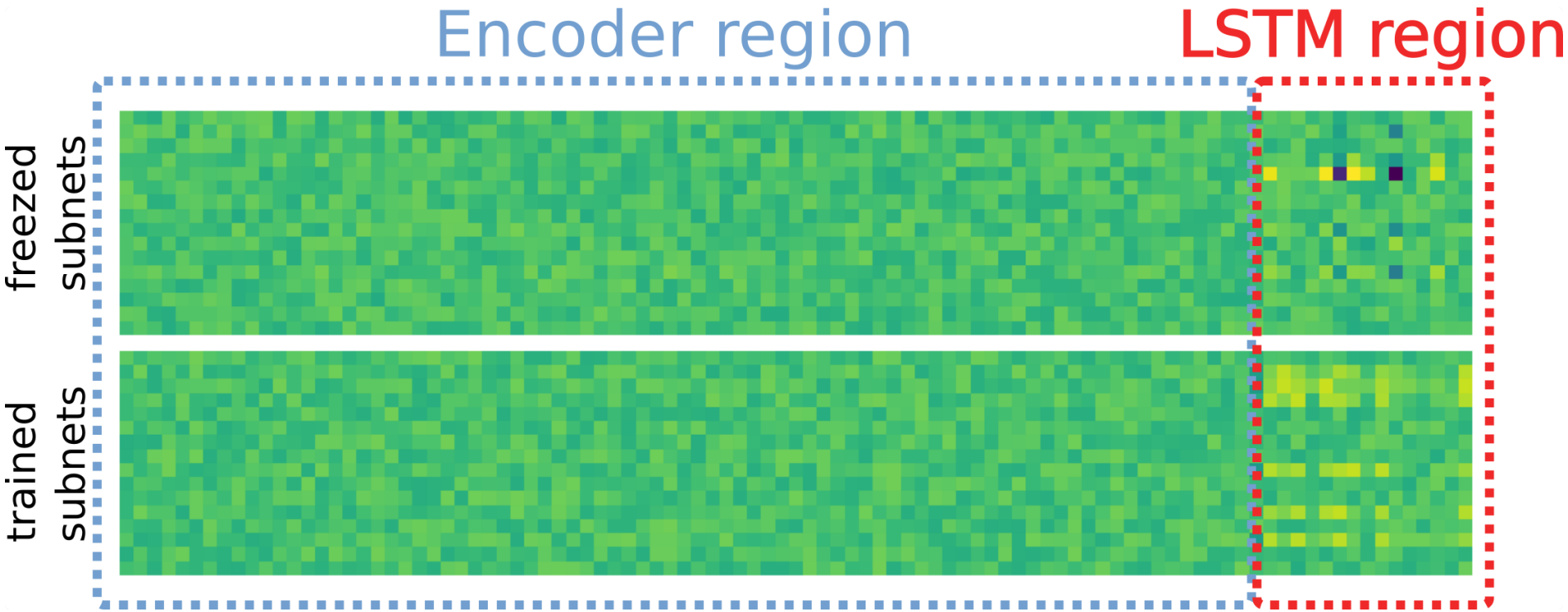


Subnets train freezing



Weights at concatenation layer

Is the encoder really useful?



Network's output

Whole Network performance

Curriculum learning

- Normal training
- Use the trained NN to score the difficulty of samples
- Schedule a training for another NN

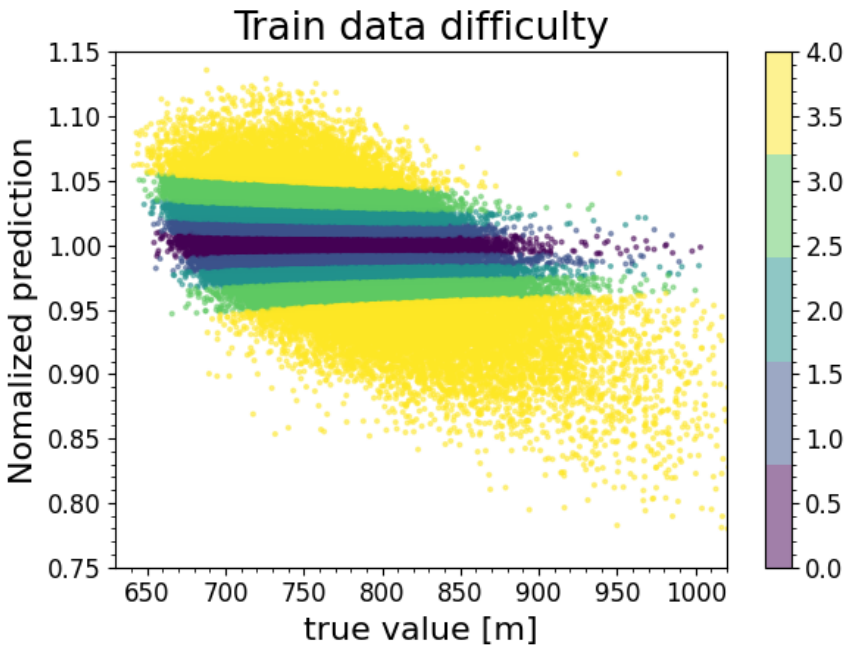
Maybe useful on classification tasks, but for regressions..

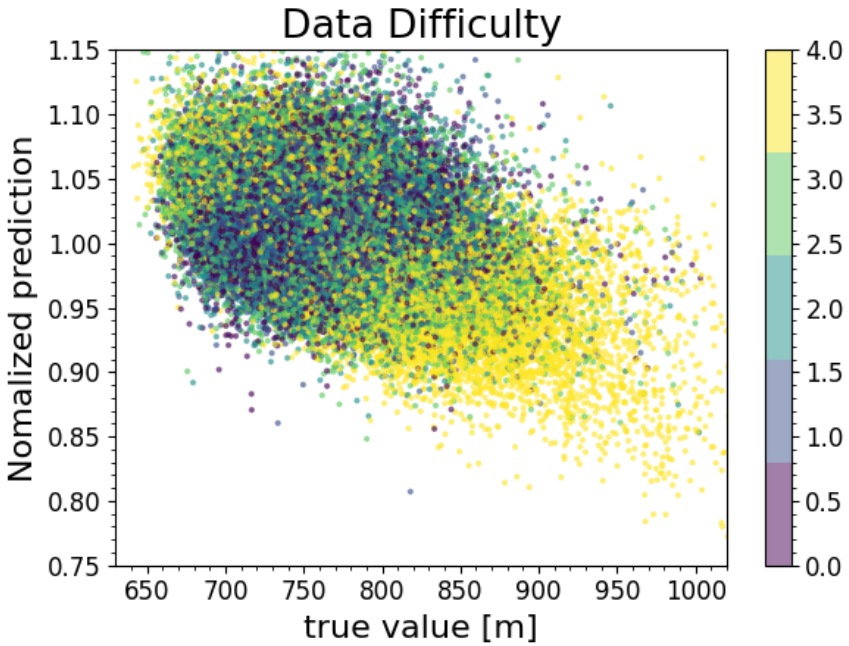
curriculum learning

Training over "difficult" data makes the net forget the easier samples

Sort in ascending error and divide in equal-length groups that the "prof" feeds to the next network

The "student" network is trained increasing in difficulty but spends more time on hard samples





Test setup on CircleCI

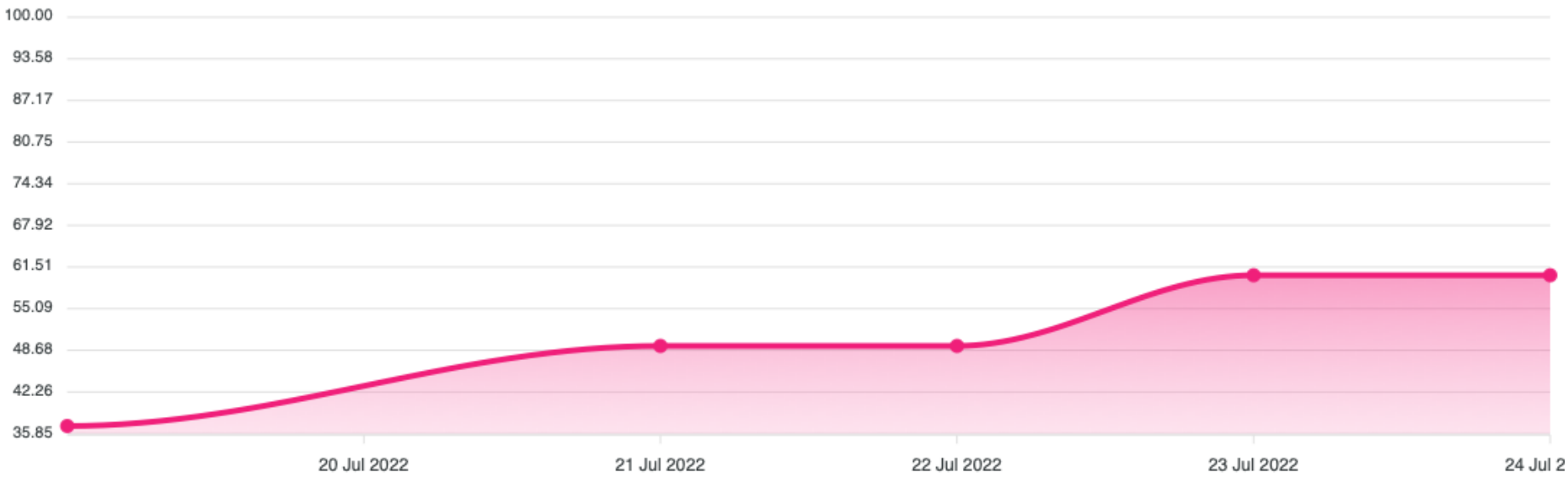
We covered $\sim 60\%$ of the whole project, main test ideas are for training classes and functions with a fititious smaller dataset.

Main tests:

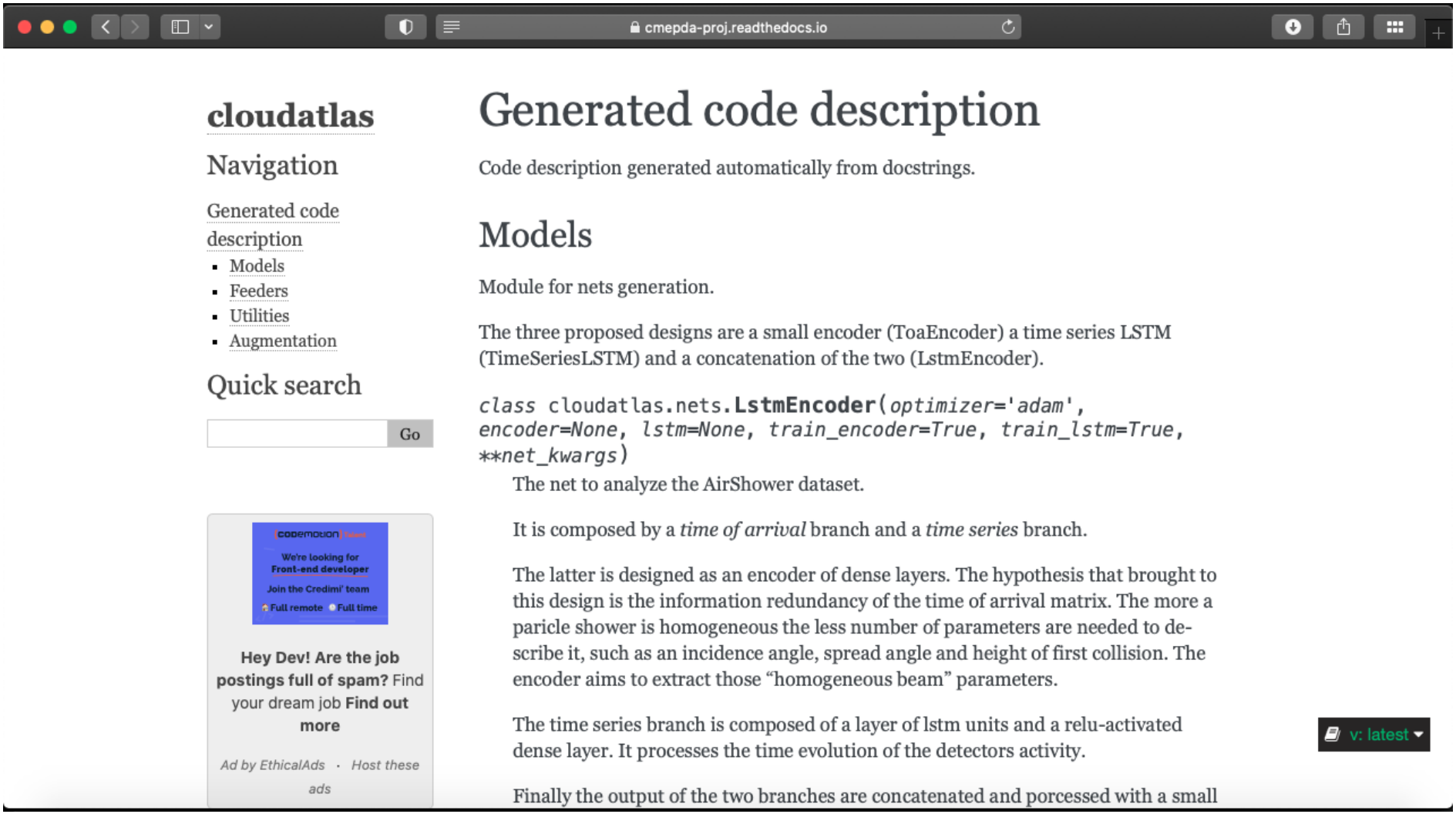
- **Net building**: checks if networks are correctly built
- **DataFeeders**: builds a fictitious dataset and checks whether `DataFeeders` work fine
- **Augmentation**: tests if an augmented matrix is effectively "rotate", "flipped", etc.

Some tests required a trained model so they were not executed in CI.

Code coverage



Documentation



Danke e bibliography

Danke Schon