EUROVOLC

European Network of
Observatories and Research Infrastructure for Volcanology

Deliverable Report

D9.3 Machine learning tool kit and database

Work Package:	*Volcano pre-eruptive unrest detection schemes*	
Work Package number:	*9*	
Work Package leader:	*DIAS C. Bean*	
Task (Activity) names:	*Joint inversion of multiparametric data + Test database construction and quality control*	
Task numbers:	*9.6 + 9.7*	
Responsible Activity leader:	*F. Cannavò*	
Lead beneficiary:	*INGV*	
Author(s)	*S. Palazzo, R. Mineo, C. Spampinato, F. Cannavò, A. Cannata, C. Montagna, J. M. Saurel, O. Geber, A. Lemarchand*	
Type of Deliverable:	*Report [x]* *Prototype [ ]*	*Demonstrator [x]* *Other [ ]*
Dissemination level:	*Public [x]* *Prog. Participants [ ]*	*Restricted Designated Group [ ]* *Confidential (consortium) [ ]*

Programme: H2020
Project number: 731070

Table of Contents

Contents

Summary

The recent success of deep learning approaches in a variety of pattern recognition fields (computer vision, natural language processing, speech recognition, game playing) and the growing availability of historical data in volcanology have posed the basis for the application of such techniques to automatic identification of patterns that lead to volcanic phenomena. However, the main limitation in investigating this line of approaches lies in the lack of annotations that describe the nature of the recorded data, preventing the use of standard supervised machine learning approaches. In this deliverable, we developed a tool that performs anomaly detection in volcanic historical data in an unsupervised way, leveraging the hypothesis that normal activity dominates the event distribution and can therefore be treated as the main source of information for training a model able to identify deviations as anomalies. These anomalies could be attributed to changes in the volcano state. We validated the effectiveness of the proposed approach on data series from Etna and Stromboli, with promising results that confirm how this approach is able to identify anomalies in the data, often with a significant anticipation period from volcanic event occurrences.

1. Introduction

Monitoring volcanic activity and predicting eruptive events is one of the main challenges in volcanology, pertaining to both the scientific perspective and the safety of a population. Unfortunately, predicting volcanic eruptions is a complex task, for which a general solution has not been found yet. Typical approaches aim at identifying "precursor" signals by monitoring geophysical/geochemical signals measured *in loco* or by remote sensing (Scarpa, 2001; Sparks, 2003; Kilburn, 2018). This kind of analysis, however, has several drawbacks. First of all, finding common patterns that precede eruptions may require manual intervention, due to the lack of completely automatic and autonomous approaches that are able to reliably extract such information. Secondly, the number of eruptive events that can be analyzed to extract this kind of information is generally limited: normal activity in a volcano is far more common than eruptive events, and the small number of events of interest - together with the psychological factor of *wanting* to find such patterns - can lead to mistakenly identify simple coincidences for discriminating patterns. Thirdly, and correlated to the limited number of events of interest, a trade-off must be chosen between how many events should be used for analysis (i.e., as source data in which precursors should be looked for) and how many should be used for validation: Including several samples for analysis increases the chances of finding robust patterns, while including several samples for validation gives more guarantees on the reliability of the proposed precursors.

These considerations motivate the need for searching for a different way to tackle this problem, data-driven techniques based on machine learning may provide a viable alternative, thanks to the availability of large datasets of volcanic activity collected through the years. However, in this case new problems arise, since most successful machine learning methods (based on the recent *deep learning* trend) employ the supervised learning paradigm, which requires that input datasets be manually annotated with a label that identifies the nature of each data sample (for example, in the task at hand, such labels could be "normal activity" or "eruptive/paroxysmal activity"). This operation is generally time-consuming and error-prone, and it is often hard to convince domain experts (as would be needed in our case) to spend their time performing this task. Moreover, it can also be difficult for experts to identify the exact times and types of eruptive events, both due to possible lack of data (e.g. absence of cameras) and in case of borderline events. Hence, carrying out supervised training on volcanic activity datasets is something that may not be feasible.

In this Task, we tackle the problem of anomaly/unrest detection in volcanic activity signals as an *unsupervised learning* task, by assuming that most recorded signals represent "normal" activity, with eruptions happening as relatively rare occurrences. Under this assumption, we train a machine learning model that is able to encode

patterns of normal activity and discover deviations from such patterns. The advantages of this approach are specular to the above-mentioned problems of traditional pattern-finding solutions: 1) no manual intervention is needed to identify patterns of interest; 2) as this type of activities dominates the event distribution, it is unlikely that coincidences arise as detected patterns; 3) since the model encodes patterns of normal activity, *all* volcanic events can be employed for validating the approach.

The proposed method employs a family of machine learning models named *auto-encoders*, that are able to compute a compact representation of an input signal by learning how to reconstruct an input signal given its encoding. A well-trained model would therefore be able to reliably reconstruct an input signal, i.e., so that a distance defined between input and output would yield a small value; however, when provided with an input signal that does not match the general data distribution employed to train the model (e.g., an anomalous signal in the geophysical/geochemical data), the computed representation would not encode information able to reconstruct the input, yielding a large distance between the latter and the reconstructed output. We exploit this property of auto-encoders as an anomaly detection approach, and identify deviations from "normal activity" as signals for which the reconstruction distance provided by the trained model is large, as described in detail in the following section.

2. Machine learning tools for anomaly detection

Machine learning is a branch of artificial intelligence that studies supervised and unsupervised learning algorithms, capable of extracting information for modeling the events described by the input data in order to make predictions. In particular, the term *deep learning* refers to a family of models that process a piece of information through a sequence of *layers*, by computing more and more semantically refined representations of the input at each step. An example architecture is shown in Fig. 1.

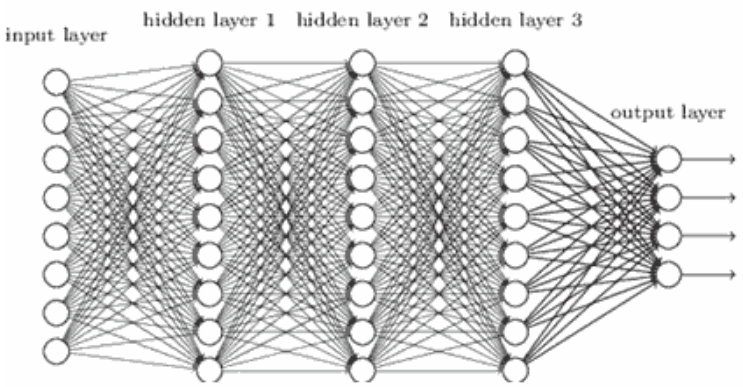


Figure 1. *Schematic representation of an artificial deep neural network.*

In the problem tackled in this Task, the need is to determine which input instances (e.g., time series of geophysical or geochemical measurements) stand out from the "normal" patterns and that represent outliers to the distribution of normal activity signals. Such instances, i.e., *anomalies*, besides being caused by errors in data due to acquisition or pre-processing problems, can be indicative of previously unknown new dynamics, and can be identified through unsupervised approaches that rely solely on intrinsic properties of input data. In this formulation of the problem, the main requirement of the data distribution is a higher prevalence of normal over anomalous instances, so that the latter represent samples that a model rarely "sees" during training and, thus, is not able to effectively learn.

Generative models, in particular autoencoders (AE) and variational autoencoders (VAE), are commonly used as anomaly detection models. Autoencoders are a type of deep architectures capable of modeling a large and multi-modal data distribution efficiently without supervision, by learning a representation that compresses input data into a space with reduced dimensionality.

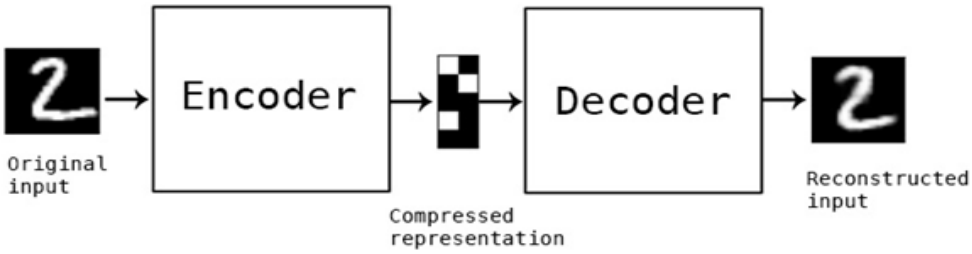


Figure 2. *Schematic representation of a generic autoencoder architecture.*

The autoencoder architecture consists of two parts (Fig. 2). The first is an *encoder* module implementing a function that reduces the size of the input to an efficient representation space without redundant information. The second part is a *decoder* that, by taking as the compressed representation vector as input, attempts to reconstruct the original signal. *Variational* autoencoders are architecturally based on the same structure, but differ in how the intermediate compact representation is modeled: in particular, VAEs learn not only a function that compresses data, but also force the compressed representation to have a standard multivariate normal Gaussian distribution. The advantage is that it is possible to sample from such distribution and synthesize new signals by feeding the sampled vector to the decoder. Figure 3 shows a diagram of variational autoencoder.

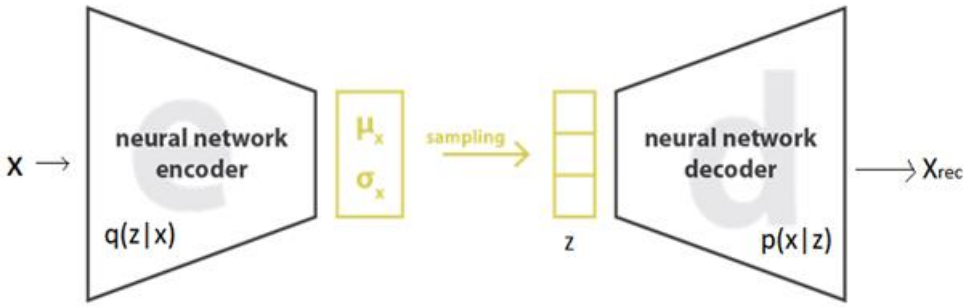


Figure 3. *Schematic representation of a generic variational autoencoder.*

Formally, it is possible to define a VAE as an encoder whose training phase is regularized to ensure that the latent space has generalizing properties. The additional regularization is the Kullback-Leibler Divergence (KLD) (Kullback & Leibler, 1951), which estimates the difference between the returned distribution and the standard multivariate normal Gaussian distribution.

3. Proposed detection approach

The approach chosen to predict eruptive phenomena is to model normal volcanic activities by detecting as anomalies or signs of unrest the signals that deviate from the reconstruction computed by the model itself. In particular, by measuring a distance between the model prediction and the real data, a probabilistic threshold for anomalous events can be established. The model processes blocks of data of size $n \times m$, where n is the number of data channels (e.g., recording stations) and m is the length of the time series segment, which is adjusted according to the availability of computing resources; 512 samples is the default value. It is worth noting that the information is taken simultaneously from several sources that are likely different in measurement type and/or geographical acquiring location. This multi-source approach can reduce false detections due to local/single failures.

The data introduced in the model undergoes the following flow:

1. **Encoder**: It is composed of a sequence of layers that process by gradually down-sampling the temporal dimension, while at the same time increasing the number of "features" (i.e., local patterns that the model

compositionally employs to represent more and more complex concepts). The first layer initially combines information from all input channels into a larger set of channels, extracting basic correlation patterns. Then, the sequence of following layers is adaptively built given a set of parameters (the initial temporal size, the minimal temporal size to down-sample, the maximum number of features), by interleaving cascades of 1D convolutional layers with down-sampling layers (implemented as 1D convolutions with stride 2).

2. **Bottleneck**: This block processes the low-resolution representation estimated by the decoder, through a further compression of the number of features, to enforce a more compact and meaningful encoding. When using a VAE model, this block estimates the mean and standard deviation associated in the latent space to the input samples, in order to apply the constraint of being Gaussian-distributed.
3. **Decoder**: This final block processes the compact representation computed by the encoder, by recovering the original temporal resolution. The architecture is symmetrical to the encoder's, with the difference that down-sampling layers are changed by up-sampling layers, implemented as 1D transposed convolutions to increase the temporal size of the signal.

In our implementation, with initial temporal size of 512 for each input channel, the model consists of 37 layers. Hyperparameters details are listed below when discussing the training configuration file.

The objective of training is the minimization of the mean square error (MSE) between the input and reconstructed signals. In the case of VAE, an error term based on the Kullback-Leibler divergence between the bottleneck output and the standard multivariate normal distribution is added to the objective function.

The proposed anomaly detection approach is based on measuring the distance L2 (Euclidean distance) between the input signal and its reconstruction. After the training phase on the portion of data of normal class, the model is expected to be able to reliably reconstruct signals that belong to the data distribution seen during training (i.e., mostly normal behavior); in the case of signals deviating from the normal behavior (e.g. during or preceding volcanic activity) the model will not know how to encode this type of signal and consequently will not be able to reconstruct it effectively. Below is a schematic of the validation process.

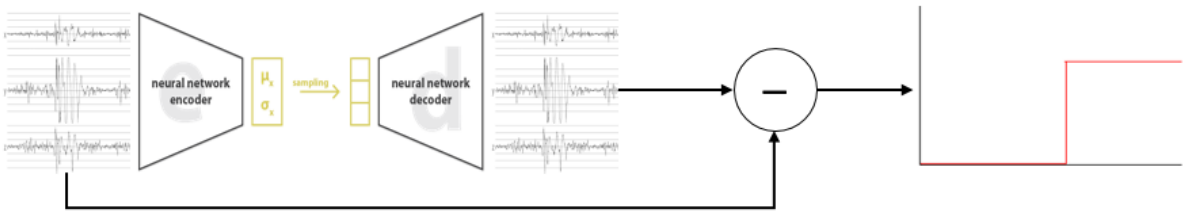


Figure 5. *Schematic representation of use of VAE for unrest detection*

4. Case of study

Dataset

For demonstration purposes, we trained and validated our VAE model on activity signals of the Etna volcano for the years 2011, 2012 and 2015. The dataset includes 20 seismic stations deployed on the Etna, originally sampled at 100 Hz, then downsampled to 50 Hz for easier handling. Each signal is provided with an expert's annotation who classifies it as normal (label 0), Strombolian (label 1) or paroxysmal (label 2) activity (Cannavo et al., 2017).

The VAE model was trained on sequences of this described dataset from the year 2011. Signals were divided into chunks of 512 samples. No data filtering was performed to avoid introducing sources of error, and because auto-encoders remove noisy components that are irrelevant to signal reconstruction. Data standardization was carried

out, and sequences classified as normal activities that had a very different scale of values from the others, probably as a consequence of instrumental errors, were removed. In the training phase the model processed only signals relating to normal activity; in the validation phase signals from 2012 and 2015 including paroxysms were used.

During the training, the ADAM optimizer (Kingma & Ba, 2014) was used with a learning rate of 10^{-5}. Figure 6 shows one of the reconstructions of a single station.

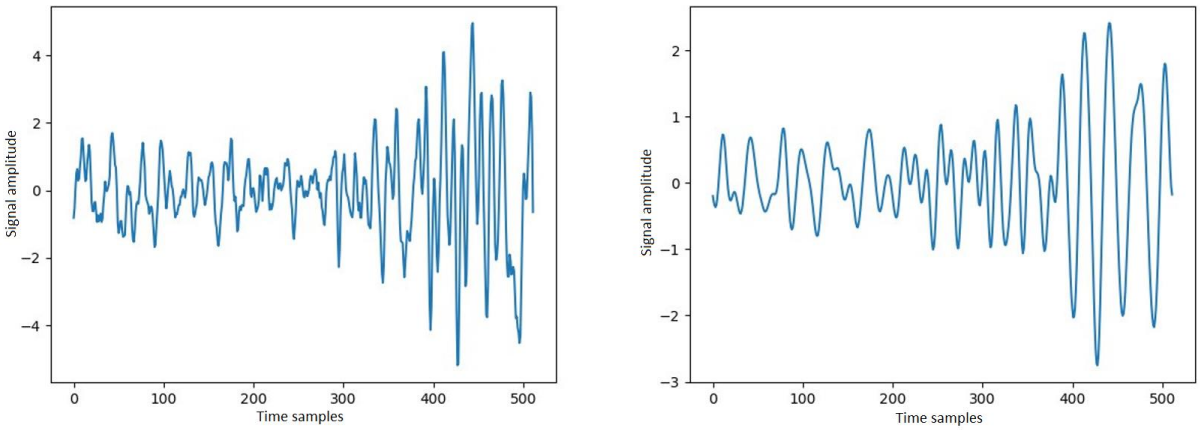


Figure 6. *Example of model reconstruction: on the left the original signal, on the right the reconstructed one*

It is evident how the model filters the high frequencies, that anyway do not play a big role in the distance calculation. The scale amplitude is not always reconstructed in an accurate way; this is easily solved by normalizing the scale of the reconstructed signal with the original one.
Figures 7 and 8 show the calculated distances from the model prediction for the periods February-March and March-April 2012 when different paroxysmal events (class 2) occurred. It is possible to note that reconstruction peaks match non-normal activity, often anticipating eruptive events.

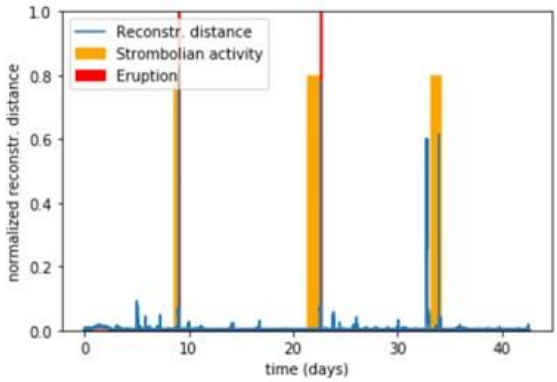


Figure 7. *Reconstruction distance of a portion of February - March 2012*

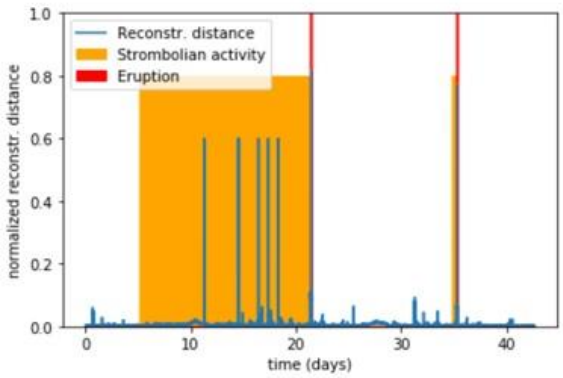


Figure 8. *Reconstruction distance of a portion of March - April 2012*

Below is the graph of the distances of a portion of the month of December 2015 during the only paroxysmal event (class 2) of the year.

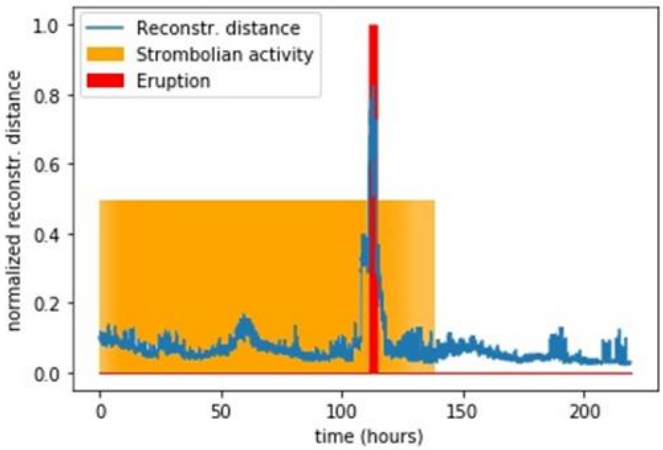


Figure 9. *Reconstruction distance of a portion of December 2015*

In order to provide a quantitative evaluation of the capability of the proposed approach to distinguish between normal and anomalous activity, we plot the ROC curve obtained when using the reconstruction distance as a classification threshold between normal signals (label 0) and joint Strombolian/eruptive activity (labels 1 and 2). The resulting ROC curve is shown in Fig. 10, with a corresponding AUC (area under the curve) value of 0.88, demonstrating a satisfactory capability to distinguish between the two groups of events.

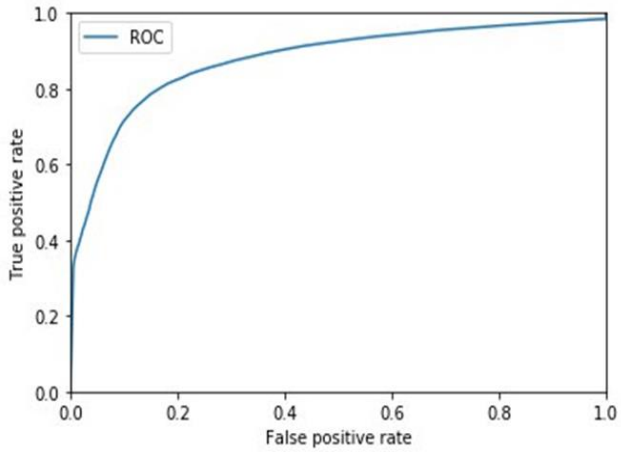


Figure 10. *Receiver Operating Characteristic (ROC) curve to evaluate the performance of the proposed scheme in detecting anomalous signals in geophysical data associated with volcanic activity.*

5. The VolUnD Toolkit (VOLcano UNrest Detection)

In this deliverable, we also present the toolkit that we developed, that can be used on arbitrary monitoring datasets. The toolkit is developed in Python 3 using the PyTorch library (https://pytorch.org/), and is structured as follows.

The root directory contains the following folders:

- **cache (internal use)**: storage directory for internally-processed dataset.

- **dataset**: directory containing default locations for train/validation/test files. Each directory can contain an arbitrary number of files, each of which must be saved in PyTorch format (using torch.save) in dictionary format, containing the following keys
  - CHANNEL_NAMES: list of names for each channels,
  - TIME_DESC: natural-language description of the temporal interval represented in the file
  - DATA: float tensor of size "stations × number of signals × chunk length
  - LABEL (optional): 0 for no activity or normal activity, 1 for e.g. mild volcanic activity, 2 for e.g. energetic eruptive activity; if not provided, non-normal events will not be emphasized during visualization
  - 'TIMESTAMP: list of Unix timestamps of size "number of signals", corresponding to the signals in DATA
- **logs**: directory where training sessions are saved.
- **utils (internal use)**: directory containing the main source code.

The main files in the toolkit are:

- **training.py**: starts the training phase; a web dashboard will also be launched where it is possible to monitor training progress through various plots.
- **visualization.py**: starts an instance of the backend to view past training sessions on the web dashboard sessions.
- **testing.py**: shows reconstruction distances on test data.
- **trainingSetup.txt**: configures the training options. Each option is specified in a single line, using the following syntax: `key: value`
  where "key" is an option name, and "value" is the corresponding value. String values should be quoted; numeric values should not be quoted; unspecified values can be provided as "None" (unquoted); list values can be grouped between brackets.
  Possible options are:
  - *train_dir*: folder where the dataset files for the training phase are located
  - *val_dir*: folder where the dataset files for the validation phase are located
  - *data_len*: chunk length (i.e., temporal length of a single input to the model); default 512
  - *channels_list*: if present, list of channels (i.e., input stations) to use; default "None"
  - *batch_size*: mini-batch size for gradient descent; default 128
  - *data_provider*: specifies whether data should be stored on RAM (faster; value "ram") or should be read from the filesystem (slower; value "fs"); default "ram"
  - *mean*: if not None, list of per-channel means for standardization; default None
  - *std*: if not None, list of per-channel standard deviations for standardization; default None
  - *tag*: name to assign to the training session in the web dashboard
  - *log_dir*: folder where to save the training data
  - *plot_every*: defines how often (number of iterations) dashboard figures (inputs, reconstructions) should be updated; default 1000
  - *log_every*: defines how often (number of iterations) dashboard plots (loss, accuracy) should be updated; default 10
  - *save_every*: defines how often (number of epochs) the model should be saved; default 10
  - *layers_base*: in the model, number of convolution layers to be applied before down-sampling or up-sampling; default 1
  - *channels_base*: in the model, initial number of channels computed from the input signal; default 16
  - *min_spatial_size*: in the model, minimum temporal size (in spite of the name), under which down-sampling should not be performed; default 2
  - *start_dilation*: in the model, initial dilation values in the encoder's convolutional layers; default 3

  - *min_sig_dil_ratio*: in the model, minimum ratio between temporal length of the signal at each layer and the corresponding dilation value; which the ratio is smaller, dilation is reduced; default 50
  - *max_channels*: in the model, channels are doubled at each down-sampling or up-sampling layer, until the maximum number of channels is reached; default 1024
  - *h_size*: in the model, size of the representation at the bottleneck; default 64
  - *enable_variational*: choose whether to use AE (False) or VAE (True); default False
  - *optim*: optimizer to use; default Adam
  - *reduce_lr_every*: defines how often (number of epochs) learning rate should be reduced; default None
  - *reduce_lr_factor*: defines the factor by which the learning rate should be reduced; default 0.1
  - *weight_decay*: weight for L2 regularization; default 0.0005
  - *resume*: checkpoint folder to continue previous training
  - *epochs*: number of total training epochs; default 32000
  - *lr*: starting learning rate; default 0.00001
  - *device*: processor to use for training (cpu or cuda); default "cuda"
- **visualizationSetup.txt**: to configure visualization parameters. Parameters:
  - *logs_dir*: folder where to find the previously saved training sessions
- **testingSetup.txt**: to configure the testing parameters. Parameters:
  - *checkpoint*: model to be validated (it can be a specific checkpoint file or the folder containing all checkpoints; in this case, the best checkpoint based on training loss will be selected)
  - *test_dir*: folder where the dataset files for the testing phase are located
  - *data_len*: as above
  - *channels_list*: as above
  - *batch_size*: as above
  - *data_provider*: as above
  - *device*: as above
  - *mean*: as above
  - *std*: as above
- **requirements.txt**: can be used to install the modules necessary for the correct functioning of the toolkit by running the following command: "*pip install -r requirements.txt*"

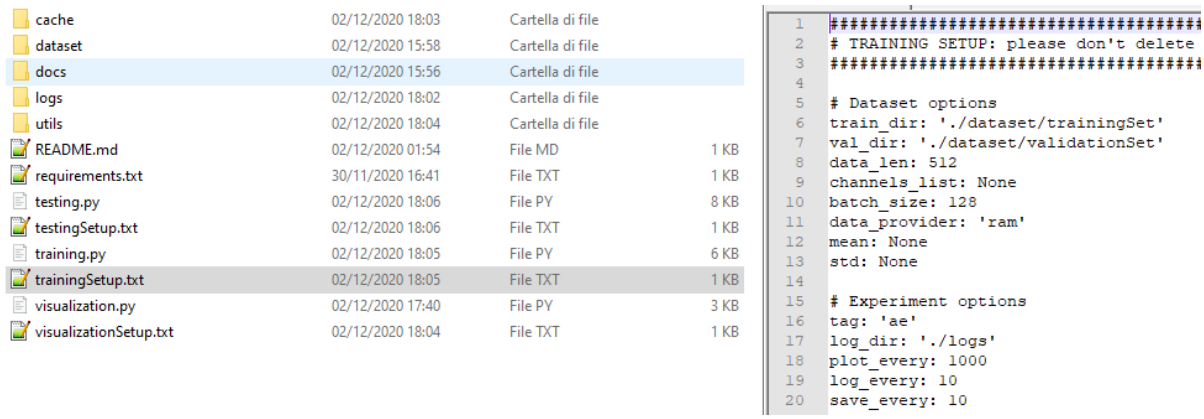
The toolkit is openly accessible through a GitHub repository at https://github.com/EUROVOLC-ML/VolUnD-Toolkit

6. VOLUND Tutorial

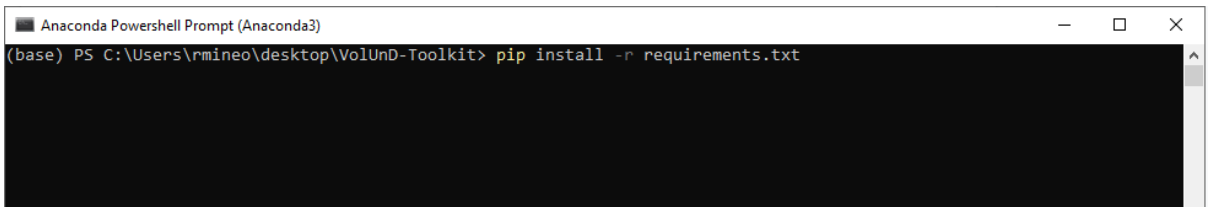
Let's assume that training, validation and test files are located within `dataset/trainingSet`, `dataset/validationSet` and `dataset/testSet`, respectively.

cache	02/12/2020 18:03	Cartella di file	
dataset	02/12/2020 15:58	Cartella di file	
docs	02/12/2020 15:56	Cartella di file	
logs	02/12/2020 18:02	Cartella di file	
utils	02/12/2020 18:04	Cartella di file	
README.md	02/12/2020 01:54	File MD	1 KB
requirements.txt	30/11/2020 16:41	File TXT	1 KB
testing.py	02/12/2020 18:06	File PY	8 KB
testingSetup.txt	02/12/2020 18:06	File TXT	1 KB
training.py	02/12/2020 18:05	File PY	6 KB
trainingSetup.txt	02/12/2020 18:05	File TXT	1 KB
visualization.py	02/12/2020 17:40	File PY	3 KB
visualizationSetup.txt	02/12/2020 18:04	File TXT	1 KB

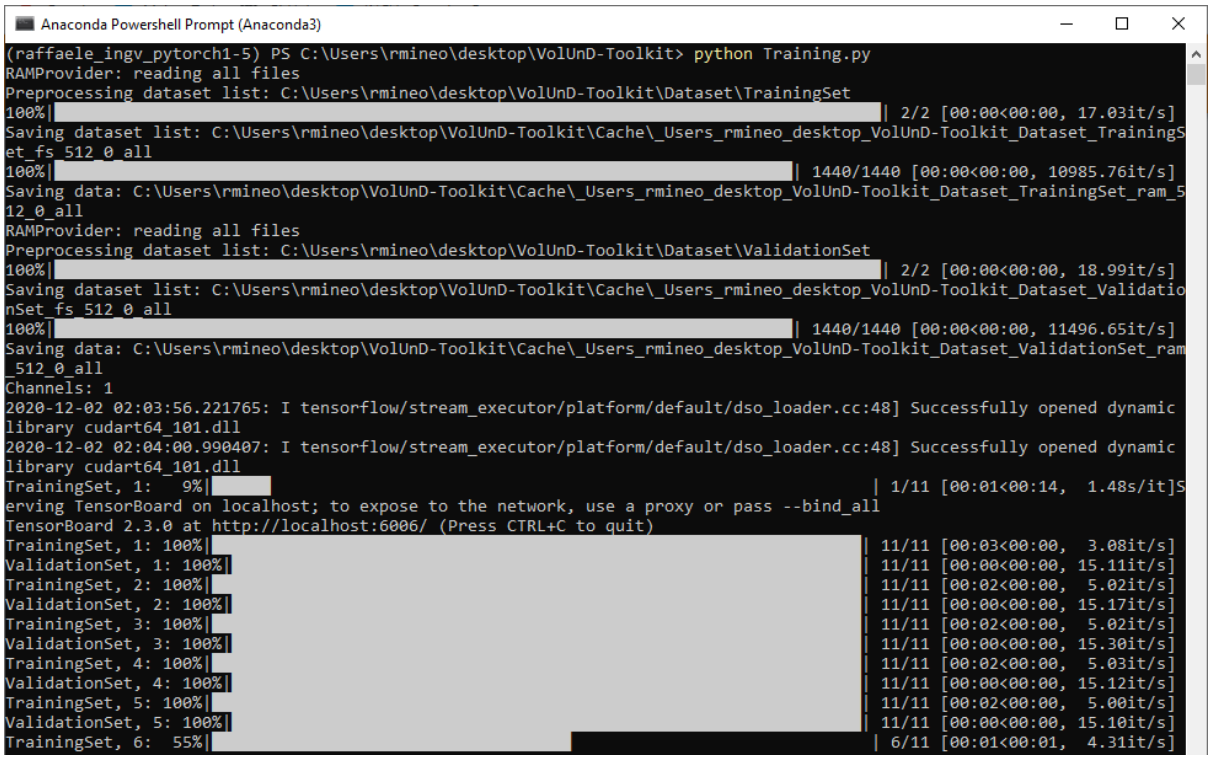
Training parameters can be configured by editing the `trainingSetup.txt` file. In particular, different directories for training and validation data can be specified.



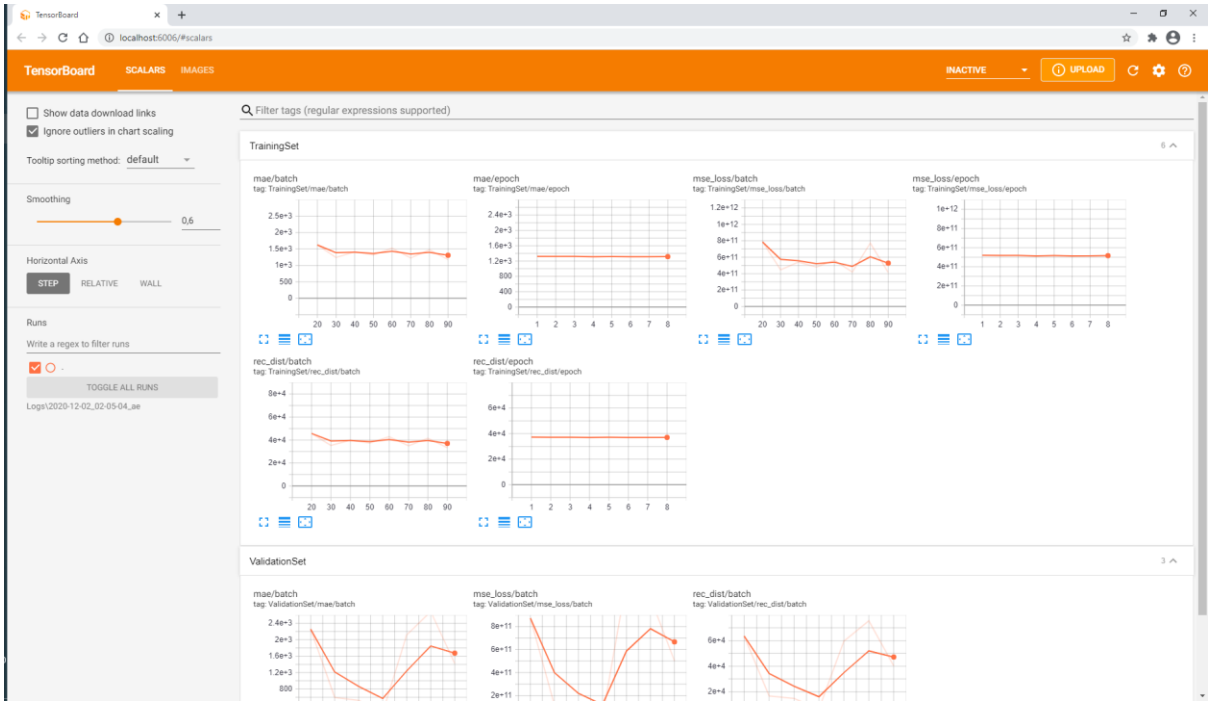
Required libraries can be installed using `pip` and the provided `requirements.txt` file. We suggest that an isolated Python environment be set up for running the experiments, using tools such as `virtualenv` or `conda`.

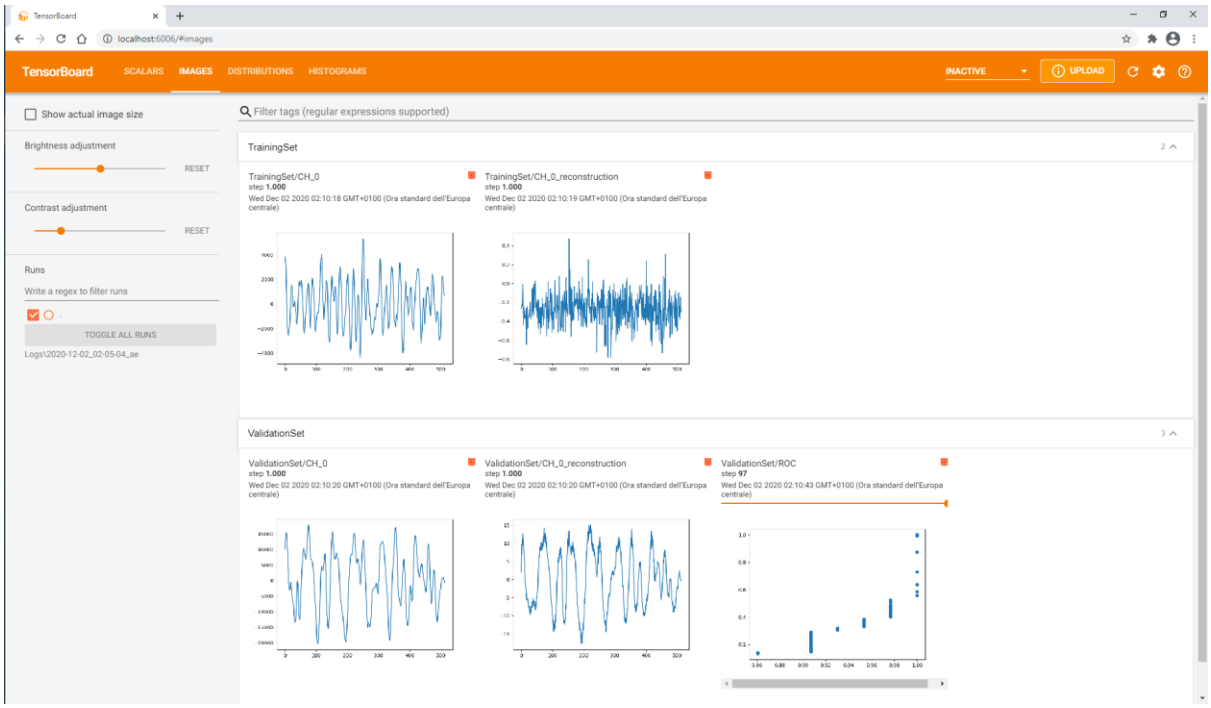


At this point, training can be run, as follows. Progress information is shown during training.

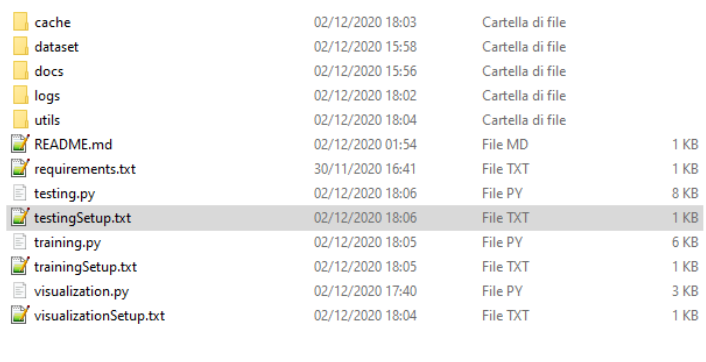


For visualization purposes and training monitoring, the TensorBoard tool will be run. The tool should open automatically on a browser window; however, it can be manually accessed by typing `localhost:6006` on the browser's address bar.

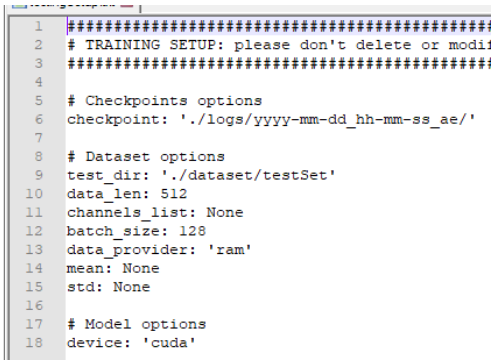




When training completes, test analysis can be carried out. The model to be tested should be specified in the `checkpoint` parameter of the `testingSetup.txt` file.

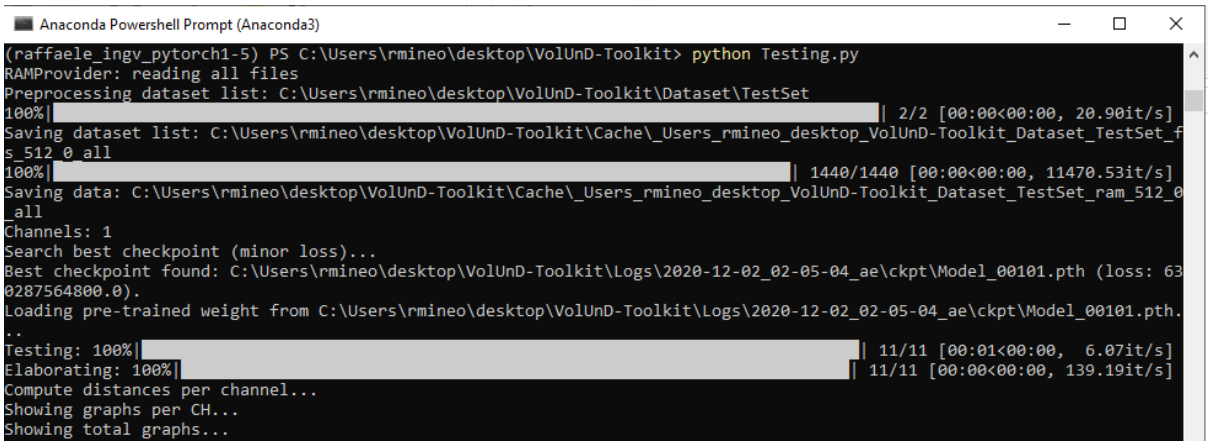


cache	02/12/2020 18:03	Cartella di file	
dataset	02/12/2020 15:58	Cartella di file	
docs	02/12/2020 15:56	Cartella di file	
logs	02/12/2020 18:02	Cartella di file	
utils	02/12/2020 18:04	Cartella di file	
README.md	02/12/2020 01:54	File MD	1 KB
requirements.txt	30/11/2020 16:41	File TXT	1 KB
testing.py	02/12/2020 18:06	File PY	8 KB
testingSetup.txt	02/12/2020 18:06	File TXT	1 KB
training.py	02/12/2020 18:05	File PY	6 KB
trainingSetup.txt	02/12/2020 18:05	File TXT	1 KB
visualization.py	02/12/2020 17:40	File PY	3 KB
visualizationSetup.txt	02/12/2020 18:04	File TXT	1 KB



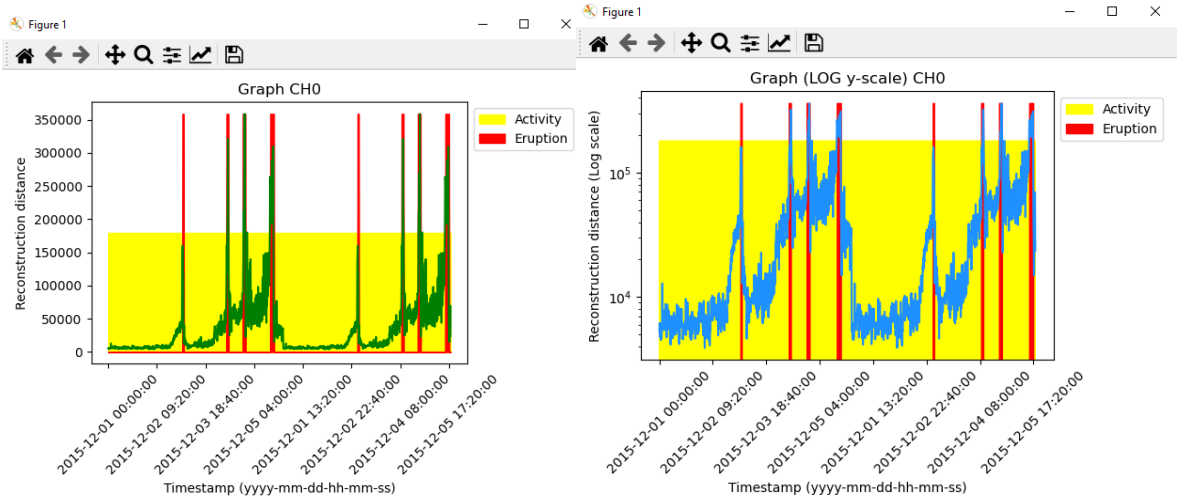
```
1  ##############################################
2  # TRAINING SETUP: please don't delete or modi
3  ##############################################
4
5  # Checkpoints options
6  checkpoint: './logs/yyyy-mm-dd_hh-mm-ss_ae/'
7
8  # Dataset options
9  test_dir: './dataset/testSet'
10 data_len: 512
11 channels_list: None
12 batch_size: 128
13 data_provider: 'ram'
14 mean: None
15 std: None
16
17 # Model options
18 device: 'cuda'
```

Visualization of reconstruction distances computed for test data can be started as follows:

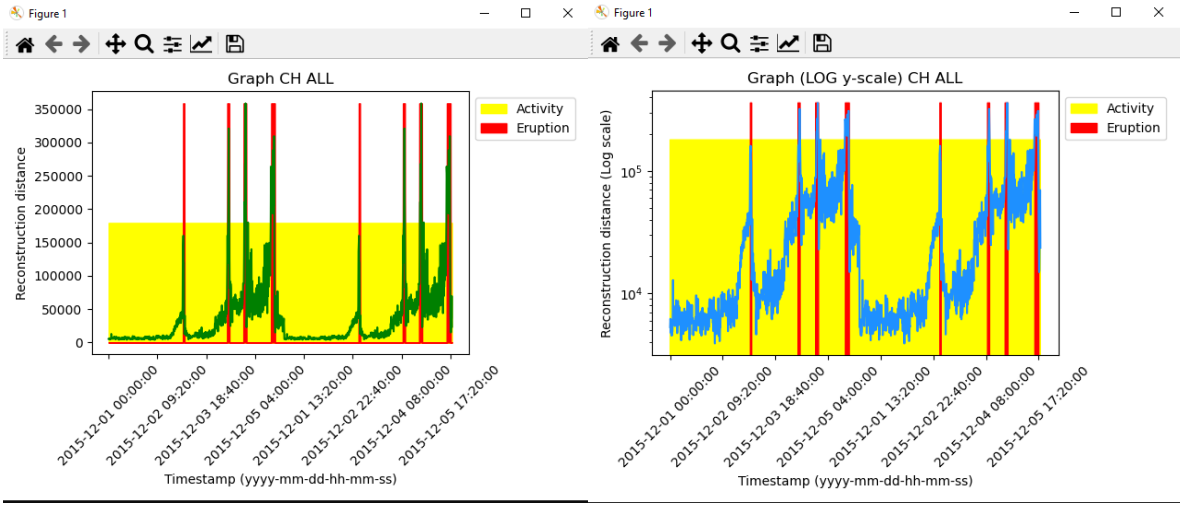


```
Anaconda Powershell Prompt (Anaconda3)
(raffaele_ingv_pytorch1-5) PS C:\Users\rmineo\desktop\VolUnD-Toolkit> python Testing.py
RAMProvider: reading all files
Preprocessing dataset list: C:\Users\rmineo\desktop\VolUnD-Toolkit\Dataset\TestSet
100%|| 2/2 [00:00<00:00, 20.90it/s]
Saving dataset list: C:\Users\rmineo\desktop\VolUnD-Toolkit\Cache\_Users_rmineo_desktop_VolUnD-Toolkit_Dataset_TestSet_f
s_512_0_all
100%|| 1440/1440 [00:00<00:00, 11470.53it/s]
Saving data: C:\Users\rmineo\desktop\VolUnD-Toolkit\Cache\_Users_rmineo_desktop_VolUnD-Toolkit_Dataset_TestSet_ram_512_0
_all
Channels: 1
Search best checkpoint (minor loss)...
Best checkpoint found: C:\Users\rmineo\desktop\VolUnD-Toolkit\Logs\2020-12-02_02-05-04_ae\ckpt\Model_00101.pth (loss: 63
0287564800.0).
Loading pre-trained weight from C:\Users\rmineo\desktop\VolUnD-Toolkit\Logs\2020-12-02_02-05-04_ae\ckpt\Model_00101.pth.
..
Testing: 100%| 11/11 [00:01<00:00,  6.07it/s]
Elaborating: 100%| 11/11 [00:00<00:00, 139.19it/s]
Compute distances per channel...
Showing graphs per CH...
Showing total graphs...
```

The script processes the test dataset with the trained model, then shows plots of reconstruction distance (y-axis) and time (x-axis ,both in linear and logarithmic scale for each individual channel.



The same plots are visualized for "all" channels, by averaging the corresponding reconstruction distances.



7. Data base tools

The deliverable includes a generic time series tool to handle miniSEED format and build a multiparameter database from different types of data and different volcanoes.

The first part of the work on this task consisted of writing down and translating into English the quality control procedures developed at IPGP for seismic data and metadata and to package the verification used in order to share with the other participants.

Overview of the validation process

The validation is a process of four steps. Firstly, the metadata is analyzed alone to detect inconsistencies. Secondly the data structure (SDS) and miniSeed data headers are systematically checked. Thus, the third step consists of selecting a subset of earthquakes detected by global seismic networks to compare them to synthetic events

computed with the SCARDEC algorithm. Finally, the last step calculates a spectrum analysis (PSD - Power Spectrum Density) with a tool called PQLX.

This step by step approach guarantees seismic data and metadata in SEED formats that are clean and with verified metadata. Some steps require tools that are not included in this package, but are available elsewhere (Scardec moment tensor inversion, PQLX).

Automated software package

IPGP has undertaken the development of a new package to partly replace those early scripts with the same goal of data validation. A first beta release of this work has been done and delivered on the IPGP github repository (https://github.com/ipgp/morumotto).

This software package uses modular plugins and takes data from FDSN miniSEED (mseed) standard format. From one or several data sources (real-time, digitizer, USB key), data are verified, merged, cleaned of overlap and gaps filled if data is available. The data are finally stored in a clean, miniSEED SeisComP Data Structure (SDS) archive on disk. The tool also displays some quality measures (gaps, overlaps) on the output data archive and can process data iteratively. The flow chart for the procedure is shown below.

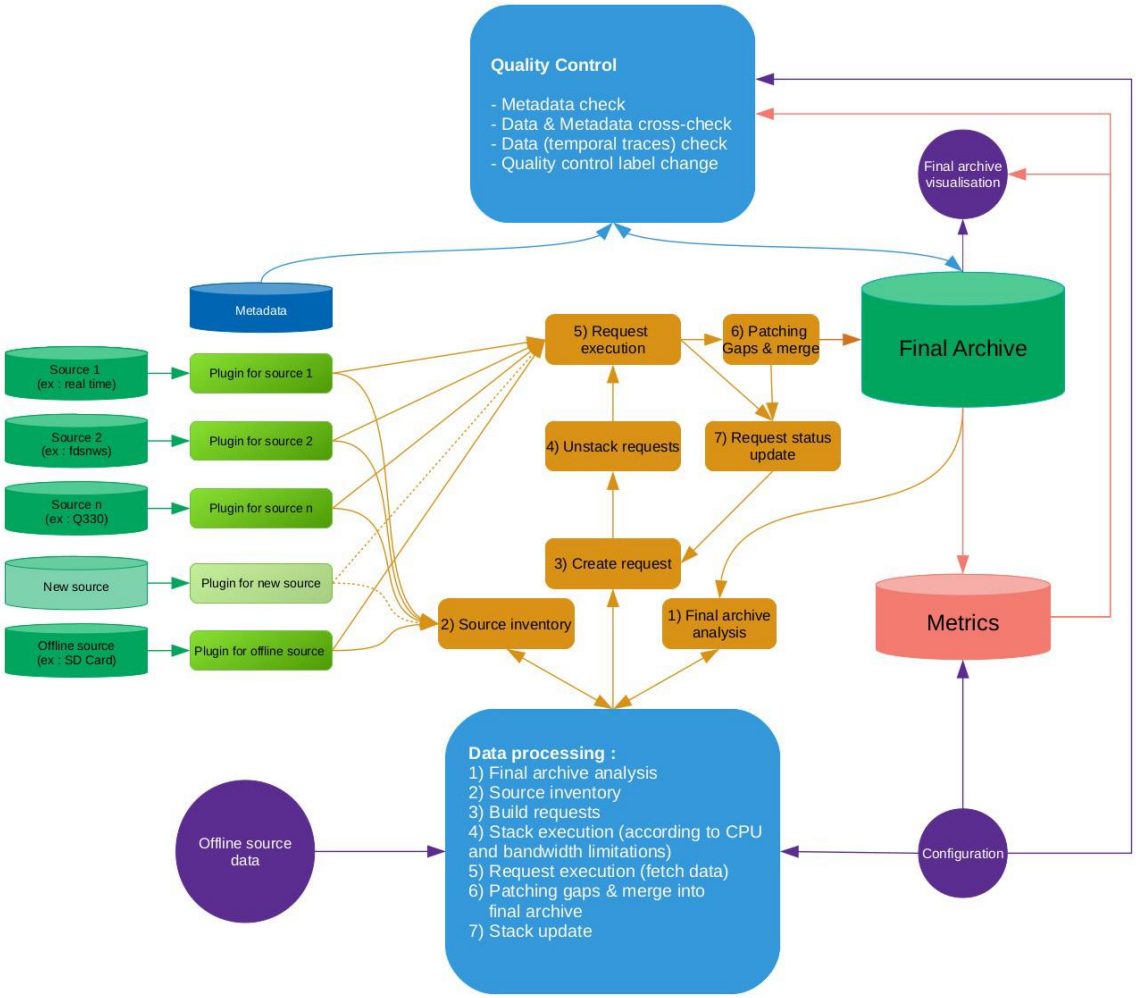
Highlights:

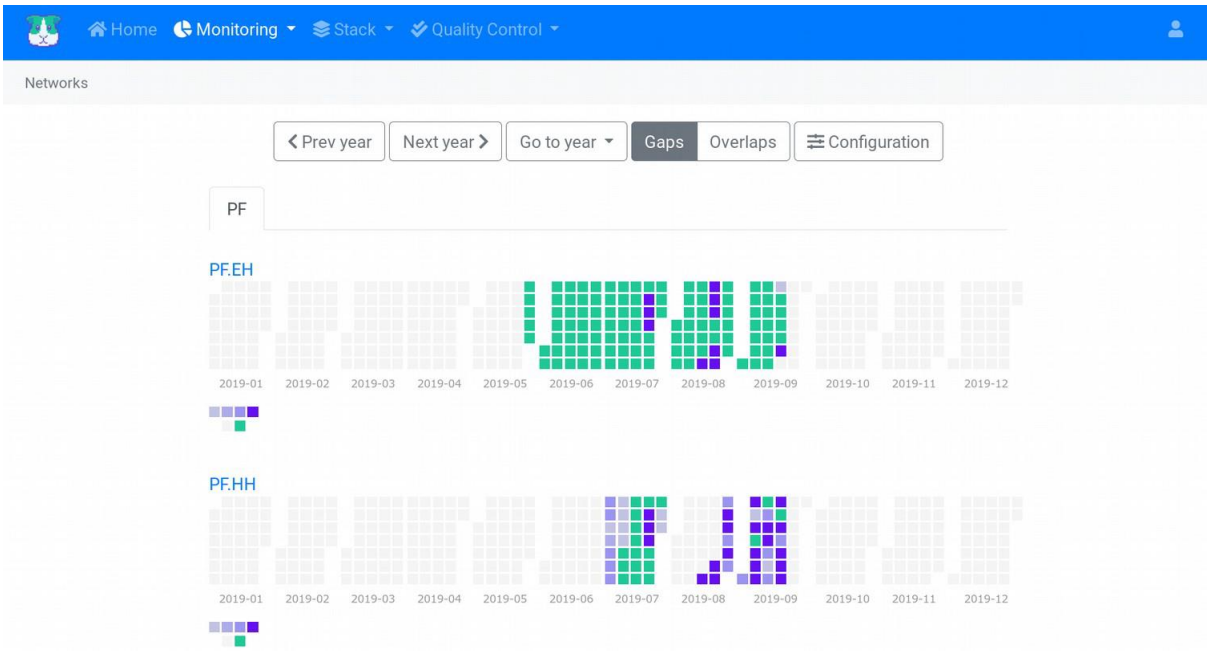
- The IPGP step by step seismic data and metadata validation guide was translated into English.
- The working environment, including scripts and tools, was packaged in a zip file.
- The scripts uses IRIS miniseed tools that are available on IRIS public github.
- The first beta release of the morumotto package has been delivered on IPGP github.
- Those three items (validation guide, validation scripts and morumotto) can be used to build a high-quality miniSEED database which can host any geophysical, regularly sampled time series supported by the FDSN standard.

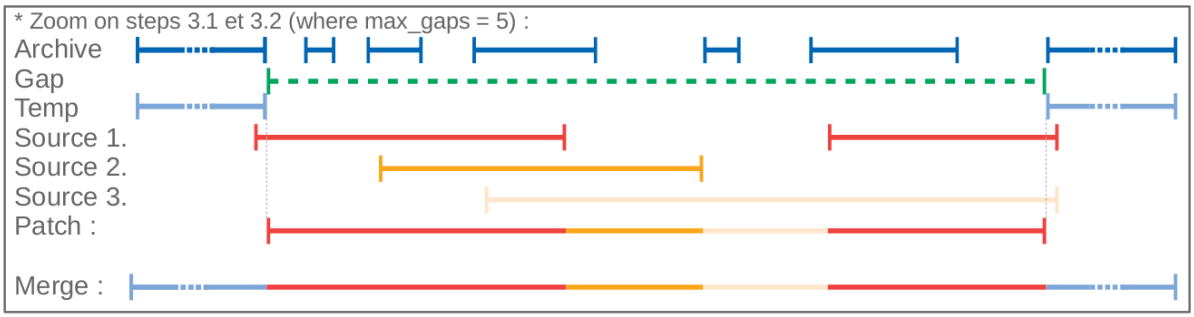
Generic timeseries in miniSEED

- The latest SeisComP3 now handles correctly any kind of miniSEED encoded timeseries (tide gauge, temperature,
- meteorological sensors, inclinometers)
- Data and metadata can then be made available through FDSN standard webservices (http://www.fdsn.org/webservices/)
- Tools are available to convert ASCII timeseries to miniSEED
- Morumotto (https://github.com/IPGP/morumotto) can be used to remove any overlaps and make sure compression and format complies with FDSN standards

Morumotto : a GUI tool to produce clean mSEED databases
Modular : plugins for different data sources (digitizers, webservices)
Not limited to seismic data, but works with any timeseries compatible with the SEED format


Quality Control
- Metadata check
- Data & Metadata cross-check
- Data (temporal traces) check
- Quality control label change
Final archive visualisation
Metadata
Final Archive
5) Request execution
6) Patching Gaps & merge
Source 1 (ex : real time)
Plugin for source 1
Source 2 (ex : fdsnws)
Plugin for source 2
Source n (ex : Q330)
Plugin for source n
New source
Plugin for new source
Offline source (ex : SD Card)
Plugin for offline source
4) Unstack requests
7) Request status update
3) Create request
2) Source inventory
1) Final archive analysis
Metrics
Data processing :
1) Final archive analysis
2) Source inventory
3) Build requests
4) Stack execution (according to CPU and bandwidth limitations)
5) Request execution (fetch data)
6) Patching gaps & merge into final archive
7) Stack update
Offline source data
Configuration


Home
Monitoring
Stack
Quality Control
Networks
Prev year
Next year
Go to year
Gaps
Overlaps
Configuration
PF
PF.EH
PF.HH
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06
2019-07
2019-08
2019-09
2019-10
2019-11
2019-12



References

- Cannavo' F. et al. (2020, May). Unsupervised deep learning on seismic data to detect volcanic unrest. In EGU General Assembly Conference Abstracts (p. 18631).
- Cannavo F. et al. (2017). A multivariate probabilistic graphical model for real-time volcano monitoring on Mount Etna. Journal of Geophysical Research: Solid Earth, 122(5), 3480-3496.
- Kilburn, C. R. (2018). Forecasting volcanic eruptions: beyond the Failure Forecast Method. Frontiers in Earth Science, 6, 133.
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
- Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.
- Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. The annals of mathematical statistics, 22(1), 79-86.
- LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K. R. (2012). Efficient backprop. In Neural networks: Tricks of the trade (pp. 9-48). Springer, Berlin, Heidelberg.
- Raghavendra Chalapathy e Sanjay Chawla. “Deep Learning for Anomaly Detection: A Survey”. In: CoRR abs/1901.03407 (2019). arXiv: 1901.03407. http://arxiv.org/abs/1901.03407.
- Georg Dorner. “Neural Networks for Time Series Processing”. In: Neural Network World 6 (1996), pp. 447-468.
- Ioffe S., and Szegedy, C., (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. CoRR abs/1502.03167. arXiv: 1502 . 03167. http://arxiv.org/abs/1502.03167.
- Scarpa, R. (2001). Predicting volcanic eruptions. Science, 293(5530), 615-616.
- Sparks, R. S. J. (2003). Forecasting volcanic eruptions. Earth and Planetary Science Letters, 210(1-2), 1-15.
- Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008, July). Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning (pp. 1096-1103).