User documentation – ClusterProcessor

1 About

Cluster Processor is a set of tools for processing the output of the Timepix3 detector. This detector was developed during the MediPix collaboration led by the CERN organization. Timepix3 is a hybrid pixel detector used in the Large Hadron Collider to capture the trajectories of the sets of elementary particles, also known as clusters. However, the amount of data generated in this process is too large for manual processing, which creates the need for automated tools which would help with processing and extracting information from the clusters. The extracted data can then be used for the classification of the clusters. The applications in the ClusterProcessor are the following:

- **ClassifierForClusters** is a console application that runs a given trained classifier on a selected dataset.
- **ClusterFilter** serves for selecting clusters based on their attributes for further processing.
- **ClusterViewer** provides a graphic user interface that visualizes clusters. It allows for interactive cluster visualization (one by one).
- **Description Generator** is a tool for generating the attribute file for the collection of clusters. This output file can be consequently used for training the classifiers.
- **ClassifierUI** provides a graphical interface with two main functionalities. Firstly, users can train a new classifier or re-train an existing classifier. And secondly, users can combine single-level classifiers into a multi-level cascade classifier.
- **ClassifierTrainer** is a console application with similar capabilities as ClassifierUI. It trains a classifier based on passed command line arguments.
- **ClusterExperiment** serves as a tool that performs classification experiments on a dataset of clusters. It trains multiple classifiers and evaluates their results.

In the following sections, we will show how to install the set of tools and use each of its individual applications.

2 Installation

In order to run all of the primary applications in the `ClusterProcessor` solution, the user has two options.

1. Download the archived build (for Windows x64) of the solution accessible on the link: https://github.com/TomSpeedy/ClusterProcessorExecutables. This method is suitable when you only plan to use the existing implementation.
2. Clone the solution from https://github.com/TomSpeedy/ClusterProcessor. This method is for those who would like to modify any of the tools of ClusterProcessor.

For compiling all of the tools, you will need the *.Net Framework* version 4.7. For further information about the *.Net Framework*, please visit https://docs.microsoft.com/en-us/dotnet/framework/get-started/system-requirements.

2.1 Data formats

The tools work with clusters that are traces of one particle with possible traces of its decay detected by Timepix3, as shown in Figure 3.3. Raw data from the Timepix3 detector contains information on hits, energy, time of arrival, and time over the threshold for each pixel. These pieces of information are grouped together by the Clusterer application (accessible at https://software.utef.cvut.cz/). Hence the base format of the input is **MM cluster format.** This file format consists of 3 separate files – ini, cl, and px file.

File	Format	Example

ini	[Measurement (or any string ending with a newline char)]
	PxFile=[Relative path from the parent directory of .ini to px file]
	ClFile=[Relative path from the parent directory of .ini to cl file]
		Measurement 123
		PxFile= Clusters_px.txt
		ClFile= Clusters_cl.txt

cl	[First ToA (decimal)] [Pixel Hit Count (integer 0-2^{32})] [LineOfStart in px file (integer 0-2^{32})] [Byte of start in px file (integer 0-2^{32})]	12345.647 100 5 30
px	[x coordinate of the pixel (integer 0-255)] [y coordinate of the pixel (integer 0-255)] [ToA (decimal)] [Energy (decimal)]	123 128 15540 14.235

Figure 2.1 MM format data structure

Another kind of data format we use in our applications is **JSON data format.** This format is used for these two purposes:

- capturing information about calculated attributes of a cluster, and
- storing multiple parameters for the classifier training.

The last data format we use in our applications is **csf** and **csf_support**. While these are not standardized data formats, they contain information about the trained classifiers so that we can reconstruct them completely from these files.

3 Cluster Viewer

Setup and input

Requirements – in case of compiling the project yourself (as described in Section 2):

- `Classifier for Clusters` project, which is part of the repository at https://github.com/TomSpeedy/ClusterProcessor,
- Third-party software for three-dimensional plotting `Chart Director for .Net` library accessible at https://www.advsofteng.com/download.html.

Application Dependencies:

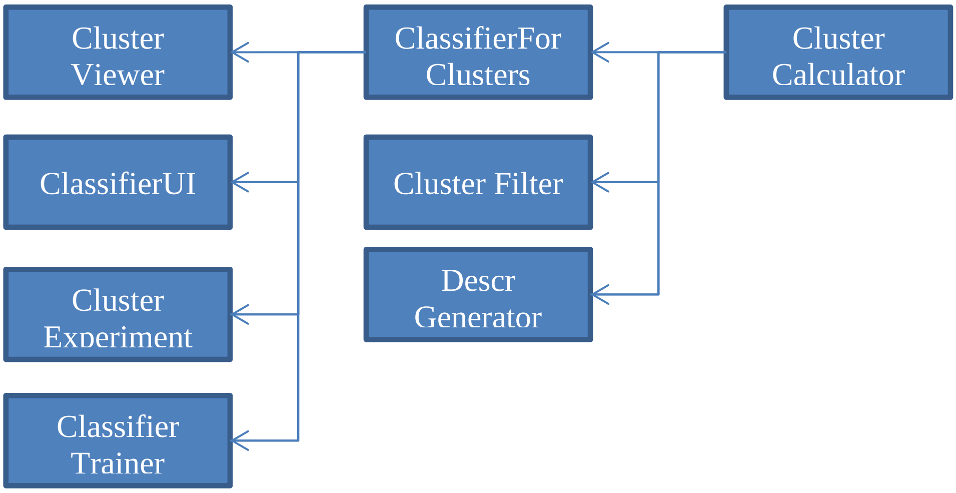
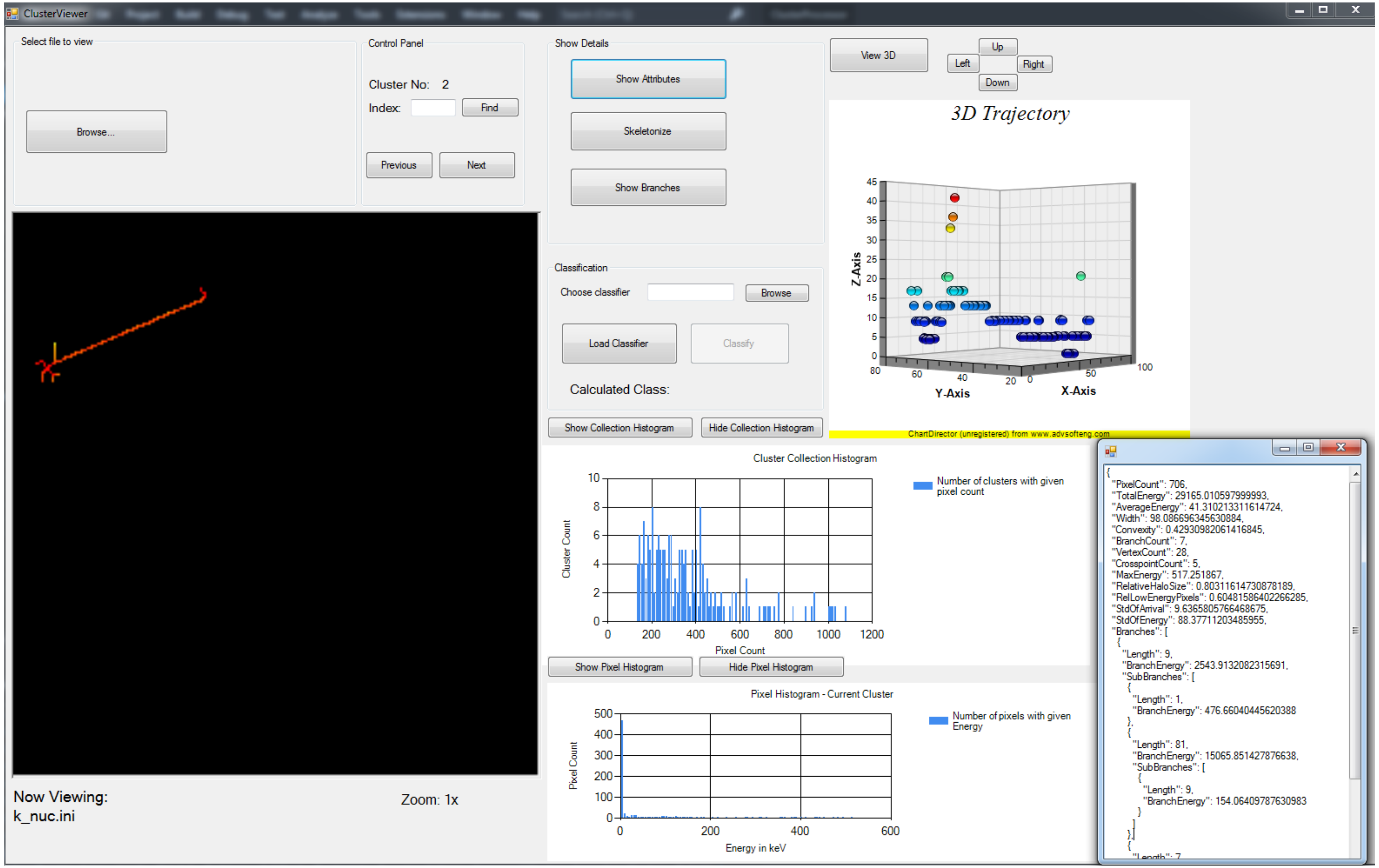


Table 3.1 Dependencies of the projects in the solution

The viewer enables displaying clusters built out of raw data from Timepix3 by the program Clusterer. However, apart from MM data format, the viewer also supports JSON data format, but only when clusters in this format have the following attributes: ClFile, PxFile, and ClIndex.



Cluster No: 2

Index:

Calculated Class:

Now Viewing:
k_nuc.ini

Zoom: 1x

```
{
 "PixelCount": 706,
 "TotalEnergy": 29165.010597999993,
 "AverageEnergy": 41.310213311614724,
 "Width": 98.086696345630884,
 "Convexity": 0.42930982061416845,
 "BranchCount": 7,
 "VertexCount": 28,
 "CrosspointCount": 5,
 "MaxEnergy": 517.251867,
 "RelativeHaloSize": 0.80311614730878189,
 "RelLowEnergyPixels": 0.60481586402266285,
 "StdOfArrival": 9.6365805766468675,
 "StdOfEnergy": 88.37711203485955,
 "Branches": [
  {
   "Length": 9,
   "BranchEnergy": 2543.9132082315691,
   "SubBranches": [
    {
     "Length": 1,
     "BranchEnergy": 476.66040445620388
    },
    {
     "Length": 81,
     "BranchEnergy": 15065.851427876638,
     "SubBranches": [
      {
       "Length": 9,
       "BranchEnergy": 154.06409787630983
      }
     ]
    },
    {
     "Length": 7
```

Figure 3.1 User interface of the ClusterViewer application

3.1 Loading a cluster file

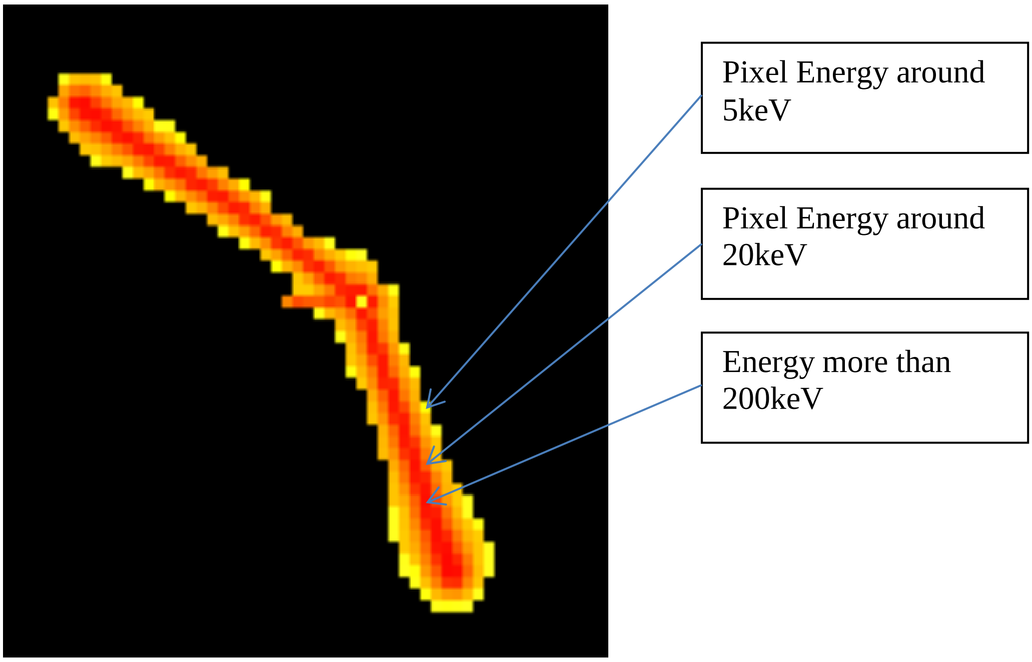
To view clusters, we run the viewer and either enter the path to our `.ini` or `.json` file or click the `Browse` button, and select the cluster file.

If the message "File was loaded successfully" appears, it means that our collection of clusters is now ready for viewing. Otherwise, a problem occurred during the loading of the file. The error is further specified in a message box. Some of the common causes are:

- `.ini` file does not exist or is inaccessible,
- `.cl` and `.px` files referenced by the `.ini` file do not exist or are inaccessible,
- `.cl`, `.px` or `.ini` file is not in the correct format.

3.2 Browsing clusters

To view a particular cluster, the user can use the ClusterViewer application, which is capable of displaying the 2D representation of the cluster. The image of the specified cluster displays the energy of each pixel logarithmically mapped to the color spectrum starting from white (for pixels with energy below 2eV) through yellow (for pixels with energy greater than 2eV and less than 15eV) to orange and red (pixels with energy up to 500eV), see Figure 3.3. If the same pixel is hit twice in the cluster, the pixel is displayed with the higher energy.



Pixel Energy around 5keV

Pixel Energy around 20keV

Energy more than 200keV

Figure 3.3 Displayed cluster color mapping

After the cluster collection is loaded, we can navigate through the collection using the `Previous` and `Next` buttons. To find the n-th cluster (clusters are numbered from 1) in the collection, we can select the index of the cluster in the `Control Panel` box and click the `Find` button.

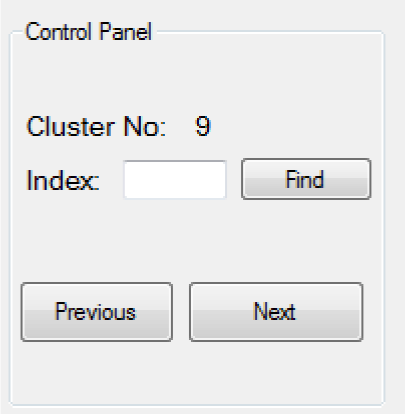


Figure 3.4 Panel for navigating through the collection of clusters

3.3 Histograms

Apart from collection browsing, users can also view histograms. There are two histograms available in the viewer:

- `Collection Histogram` displays the histogram of the whole collection for the pixel count attribute.

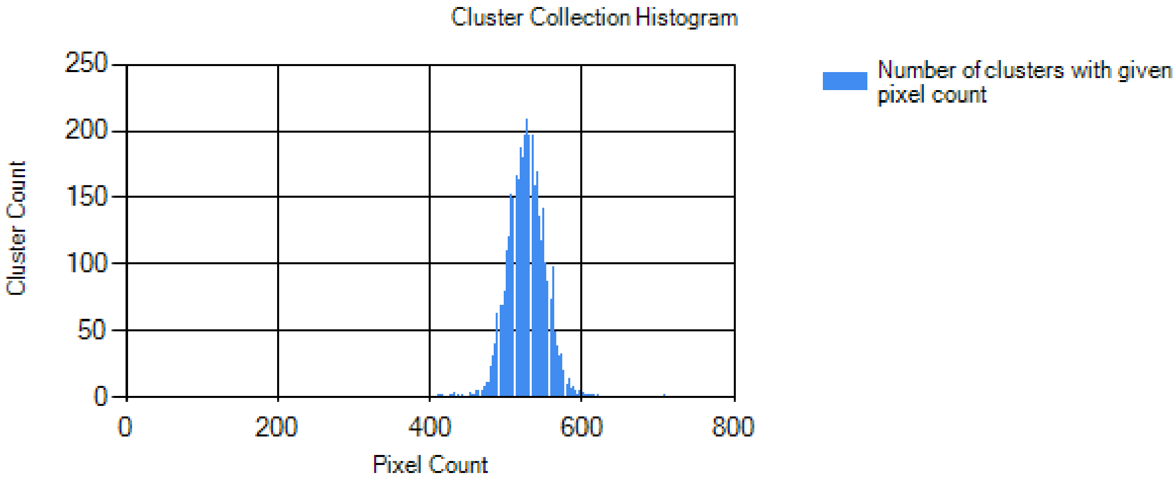


Figure 3.5 Example of a cluster collection histogram

- `Pixel Histogram` represents the histogram of the pixels of the currently viewed cluster for the energy.

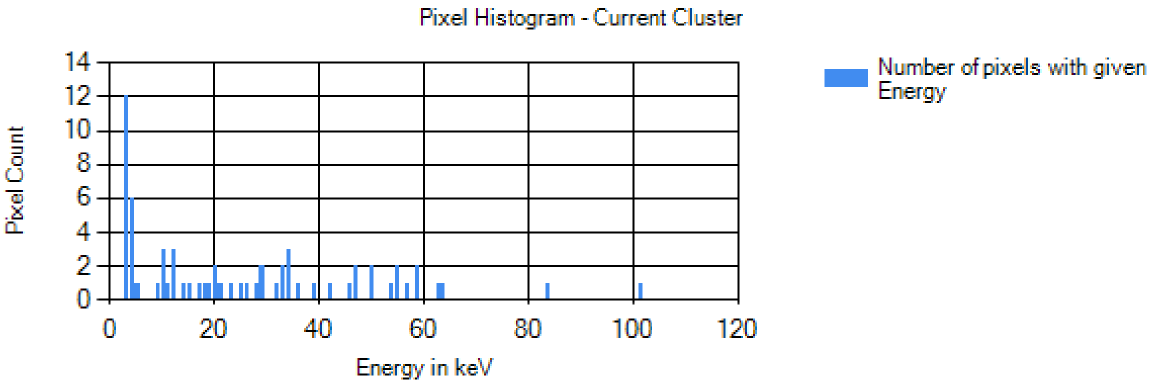


Figure 3.6 Example of a pixel histogram

Both histograms are calculated and displayed after clicking `Show Collection Histogram` (or `Show Pixel Histogram`) button.

3.4 3D visualization

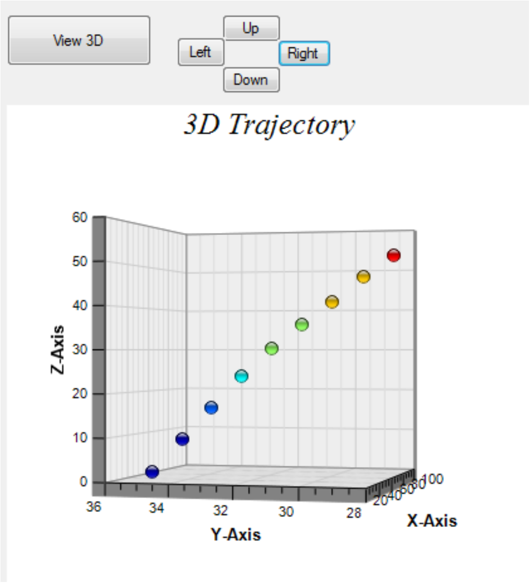


Figure 3.7 Cluster trajectory visualization

Based on the time of arrival and calibration data, it is possible to reconstruct the trajectory of particles in the cluster. To get a better idea of how the trajectory of the cluster looked like, we can visualize the cluster in 3D. The visualization is based on the ToA (time of arrival) of each pixel.

- **Viewing:** To view the three-dimensional image of the cluster, the user can click the `View 3D` button.
- **Rotation**: After the image is displayed, the user can rotate the image around the x-axis and z-axis by clicking the `Up`, `Down`, `Left,` and `Right` buttons.

3.5 Cluster Attributes

By clicking the button `Show Attributes` located in the Cluster Details section, various cluster properties are displayed. All these properties must be calculated in advance and stored in the cluster file (extension .cl). These attributes range from straightforward like `Total Energy` and `Maximum Energy` to the more sophisticated ones like `Branch Count` and `Relative Halo Size.`

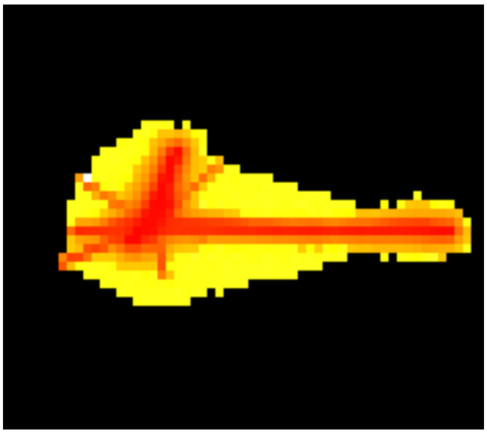


Figure 3.8 Example of a cluster with high energy (red), non-trivial halo (yellow) and branch count

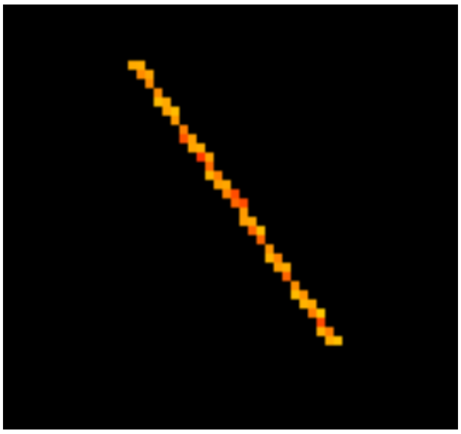


Figure 3.9 Example of a simple cluster with lower total energy. It contains only a single branch without a significant halo effect.

3.6 Skeletonization

Skeletonization is a process of transforming an image into a skeleton by reducing the original image that contains different thicknesses into a set of curves and lines. Skeletonization can be used to remove the halo effect of the clusters while preserving the shape of the original cluster. During the skeletonization process, as the image is thinning, pixel energy is split among its neighbors. This means the total energy of the cluster and its skeletonized version is the same. To view the skeleton of the original cluster, we click `Skeletonize`.

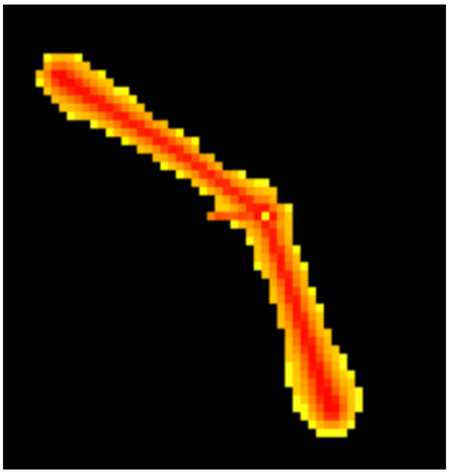
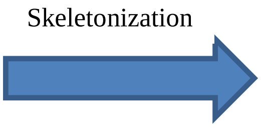


Figure 3.11 Cluster before skeletonization



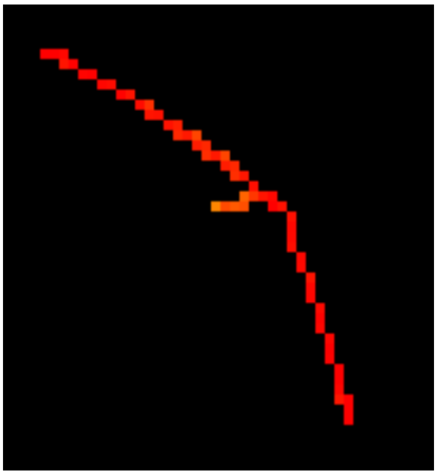


Figure 3.10 Cluster after skeletonization

3.7 Branch analysis

After finding the skeleton of the cluster, we can try to detect particle trajectories in the cluster. To do so, we can click the `Show Branches` button. The center point of the cluster is represented by the white dot, while the separate branches are denoted by distinct colors – blue, red, and green. Each branch can have its sub-branches – the starting point of the sub-branch is contained in its parent branch. The sub-branches are highlighted by a lighter shade of their parent branch color.

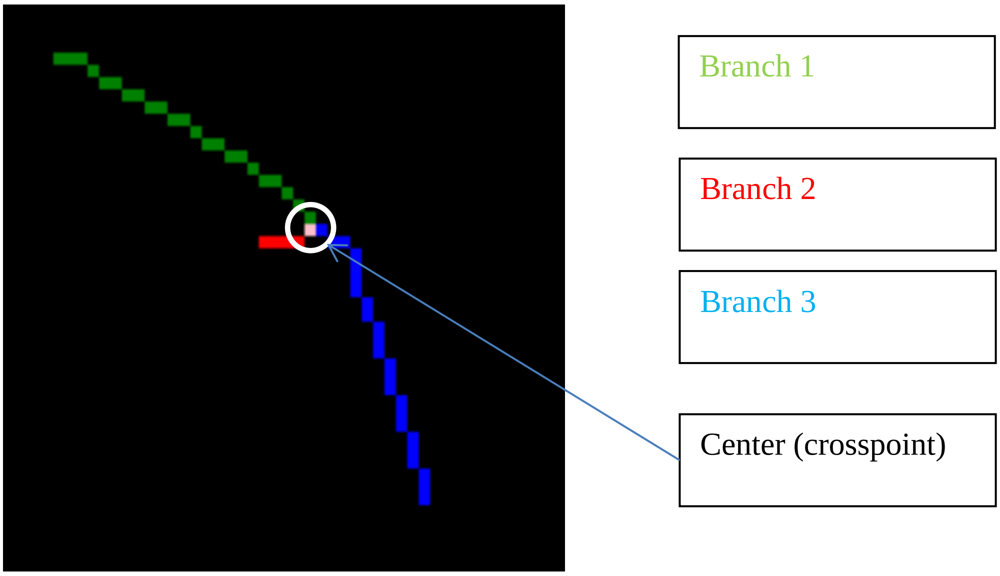


Figure 3.12 Branches of a cluster

3.8 Classification

Using all the features we are able to calculate for the cluster, we can classify the cluster into various categories. This classification is done via machine learning using neural networks. The default classes of particles that are implemented in the default `bestClassifier.csf` classifier are:

- lead,
- iron,
- helium,
- muon,

- electron,
- pion,
- low energy electron,
- and proton.

In the viewer application, a user can load a classifier by clicking `Load Classifier` and choosing the right trained classifier in `.csf` file. When the classifier is successfully loaded, it can be used to classify the currently viewed cluster.

4 Filter

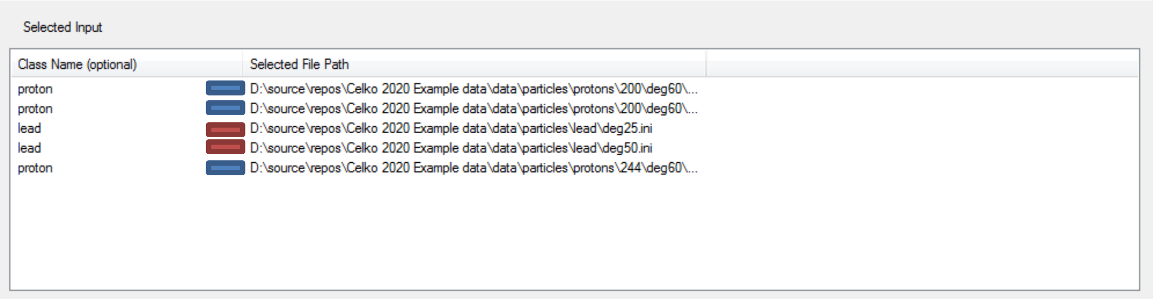
The ClusterFilter application serves as a tool for picking the clusters that fit the given criteria. The user interface of the `Cluster Filter` application is very similar to the `Cluster Viewer`.

1. As the first step, we select the input `.ini` file. This can be done either by typing in the path to the file or clicking the `Browse` button.
2. Then, we choose the name for the output `.ini` file.
3. After that, we can select the properties to filter by and set the lower and upper bounds for the property. If no bound (upper or lower) is specified, the filter automatically sets the bound to the maximum (or minimum) possible value.
4. To start the processing, we click the `Process` button. During the filtering process, a new .cl file is created. The newly created file contains only those clusters, which fit the filtering criteria. However, no `.px` file is created because the output reuses the original pixel file to speed up the calculation and prevent unnecessary copying.
5. After the filtering is done, the filter displays the message "Filtering completed successfully." If any other message is shown, it means there was an error. The message should provide more information about the problem. In most cases, it is either an incorrect data format or file inaccessibility.

5 DescriptionGenerator

The Cluster Description Generator is a tool for calculating attributes for the whole collection of clusters at once. This can be useful, e.g., when creating training data for the machine learning algorithms. The collection of clusters can be composed of clusters from several source files. The application combines clusters from several sources and produces a new collection of clusters (.ini, .cl, and .px file) containing the selected clusters together with computed attributes. The detailed instructions on how to use the tool follow:

1. We start by clicking the `Browse and Add…` button, where we select one or more `.ini` files and add them to the `Selected Input` collection.
2. To add more input files, we can repeat the process until all desired files are shown in the `Selected Input` box, as shown in Figure 5.1.
3. To remove elements in the collection, we click the `Remove Selected` button.
4. For every single file we select, we can edit its `Class Name` column by triple-clicking – there can be multiple files containing the same class. This way, we set the name of the particle present in the input file, which can then be used for the supervised machine learning algorithms.



Selected Input

Class Name (optional)	Selected File Path
proton	🟦 D:\source\repos\Celko 2020 Example data\data\particles\protons\200\deg60\...
proton	🟦 D:\source\repos\Celko 2020 Example data\data\particles\protons\200\deg60\...
lead	🟥 D:\source\repos\Celko 2020 Example data\data\particles\lead\deg25.ini
lead	🟥 D:\source\repos\Celko 2020 Example data\data\particles\lead\deg50.ini
proton	🟦 D:\source\repos\Celko 2020 Example data\data\particles\protons\244\deg60\...

🟦 “Proton“ class partitions

🟥 “Lead“ class partitions

Figure 5.1 Selected partitions in Description Generator

5. Then, we can choose the output file name in the `Select output` text field and also tick the attributes that will be calculated. To include the cluster class name as an attribute, remember to tick the `Class` attribute (which will not be actually calculated, it will just be copied from the provided class name).

6. After that, we can choose either the **even** distribution of each class in the output or the distribution **proportional** to the particular class size. Even distribution of the class means that each class has the same chance to be processed as the next cluster. In contrast, proportional distribution means the chance of picking clusters from a class with smaller datasets is smaller, while classes with bigger datasets are more likely to be picked.
7. For each file with a given class (further referenced as a class partition), we can choose whether we want to process those in a serial order (provided by the order on input – the next partition is processed after the previous was already fully processed). Another option is the **parallel order** (after a single cluster from the current partition is processed, a cluster from the next partition is set to be processed next).
8. The user can also select the **ending condition** of the process. When this condition is satisfied, the program finishes the calculation – for a large number of clusters, the process may take several minutes to complete. There are three types of ending conditions:

    a. Any partition is fully processed (**First partition**)
    b. Some class is fully processed (**First class**)
    c. All classes are fully processed (**Last class**)

By default, no cluster on the input will appear in the output more than once. When generating the data for imbalanced classes (in the real-world data, there are huge differences in class sizes), this could lead to problems with the machine learning classification of the less frequent classes. To compensate for that, we can choose the `Align class` option. By selecting the align class, any other class partition that is processed will not be discarded but will be processed repeatedly until the specified `Align class` is fully processed. If we decided to also calculate path-dependant attributes (ClFile and PxFile) we can not move the output file to any other directory – moving the file may not preserve the relative paths and make it unusable for viewing in the viewer application.

6 Classifier

The `ClassifierForClusters` is a console application that provides an interface for the classification process of the selected clusters. The prerequisite for the classifier is to have Accord.net NuGet packages installed (in case you are not using compiled executables). We need to keep in mind that

Syntax:

ClassifierForClusters.exe [trained_classifier.csf] [file_to_classify.json] [options]

Command-line options	
`--simple` or `--multi`	Specification of the classifier type (single-level or multi-level), default is single.
`--distr`	Print only key-value pairs of the class name and its frequency.
`--default`	Label each cluster with a class and print the frequencies of the classes to the console.
`--specials`	Label each cluster with a class (same as --default), print frequencies while also create a file for non-trivial unclassified clusters.
`--multiFile`	The clusters are split into separate JSON files according to the predicted class and print key-value pairs of the frequencies to the console. In order to be able to view the results in the ClusterViewer, the cluster must contain attributes ClFile, PxFile, and ClIndex.
`--singleFile`	Print all labeled clusters to a single JSON file.

Table 6.1 Options of ClassifierForClusters application

7 ClassifierUI

Classifier UI is a tool that enables training classifier models based on the input parameters. User needs to set:

- **Training Json file** = training data in JSON format,
- **Classifier config file** = parameters of the classifier in JSON format,
- **Trained model** = If we want to re-traini an existing model of single-level classifier, we fill this field with its configuration `.csf` file.
- **Minimal accuracy** = When this level of accuracy (or better) is reached on the validation set (the data which the classifier did not see during training), the training stops, and the classifier is stored into a file. It is a value between 0 and 1.
- **Maximal repetition count** = The maximal number of times we will try to train our classifier to reach the target accuracy.
- **Seed** = Integer value, based on the value of the seed, data is split randomly into training and validation sets (where 90% of the data is used for training and the remaining 10% for validation).

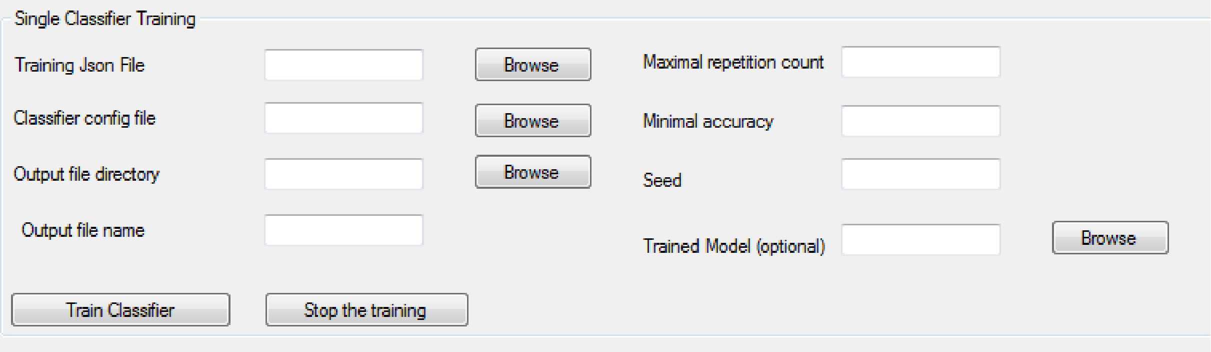


Figure 7.1 Classifier training dialog

The UI also provides an option to combine up to 4 simple trained classifiers into a multi-level classifier. Users can choose the root classifier that will be applied first. The first classifier classifies the input clusters into several classes. One of these classes can be selected as a "`Split Class`." All clusters classified into the split class by the first classifier are fed into the classifier `Trained Model Lv1` for further classification. Among the output classes of the `Trained Model Lv1` classifier, another split class can be selected, and so on. That means, for combining n classifiers, we need to fill in $n - 1$ split classes because the bottom level classifier cannot have a split class. If you use only two classifiers (root and Lvl 1), make sure all the other fields for classifiers are empty.

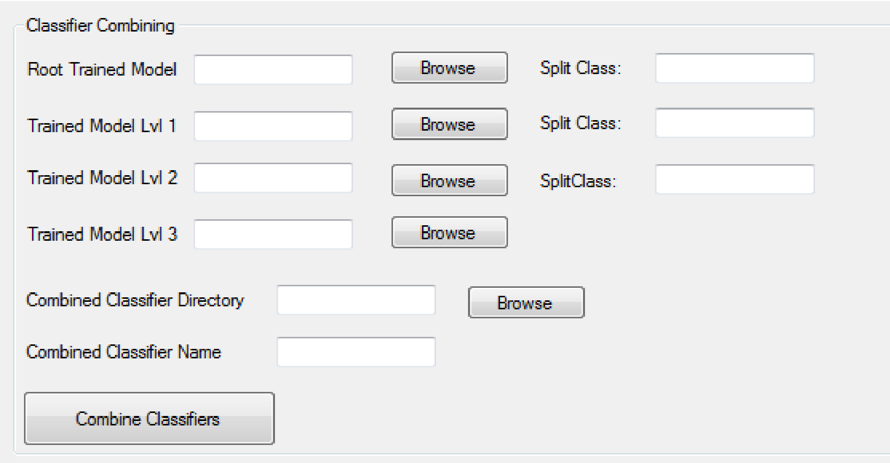


Figure 7.2 Classifier combining dialog

The example below shows a multi-level classifier that uses three simple classifiers and two split classes.

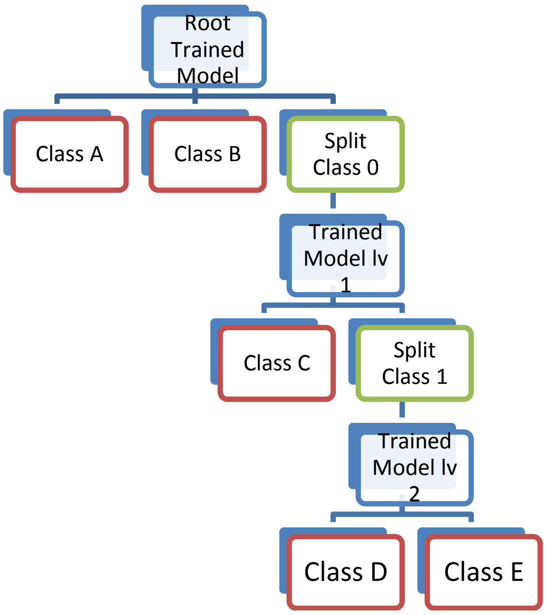


Figure 7.3 Example of a multi-level classifier

8 ClassifierTrainer

ClassifierTrainer is a console application that can train a classifier. This application can be used instead of ClassifierUI. Both tools can train the same classifier and differ only in their interfaces.

Syntax:

ClassifierTrainer.exe [training_data.json] [network_parameters_config.json] [full_output_file_name_including_path] [options followed by their value (space separated)]

Example

ClassifierTrainer.exe lead_training data.json lead_network_config_parameters.json --acc 0.8 --seed 42 --maxrep 5

Command-line options	
`--trained`	(Optional) Indicates existing .csf file should be trained again instead of creating a new classifier. A path to the trained model should follow this option.
`--acc`	Sets the target accuracy of the classifier on the validation set to save the result. Must be followed by the accuracy value between 0 and 1.
`--seed`	Sets the seed. According to this seed, the data is split into the training and validation sets. It is followed by an integer value.
`--maxrep`	Sets the maximal number of passes through the dataset. The training ends when either minimal accuracy on the validation set is achieved or the number of passes through training set reached `maxrep`.

Table 8.1 Command line parameters of the ClusterTrainer

9 ClassifierExperiment

The ClassifierExperiment is a console application for performing experiments with the classifiers with the fixed dataset. It accepts only a single parameter - a path to the directory where the training and testing data, together with trained model directories, are located. (Note: The data and models are not in the same folder - their parent directories are). The folder with the data for the experimets is accessible at https://drive.google.com/file/d/1f_t0RE7KiqO69yrzWAbU8LSX8r3l4m_J/view?usp=sharing.
Syntax:

ClassifierExperiment.exe [path to the directory that contains the train_data, test_data, and trained_models directories]

To verify that the classifier `bestClassifier.csf` was successfully loaded, it is tested on the test dataset, and the result is displayed in the console. To check the performance of each single-level classifier, we trained the following types of classifiers:

a. S_{lead} – classifies the particles into lead particles and the rest. This classifier was trained on roughly 80000 clusters.

b. $S_{fr\_he\_fe}$ – separates fragments, helium, and iron from each other and the rest. The training dataset for this classifier consists of 450000 clusters.

c. S_{le_pr} – separates low-energy electrons from protons and from the rest. The total number of clusters we used during the training is more than one million.

d. $S_{e\_m\_p\_ep}$ – splits clusters into four classes – electrons, muons, pions, and the artificial class `elPi0`. The artificial class `elPi0` contains … It is trained on the dataset of approximately 500000 clusters.

e. S_{all} – splits clusters into all categories directly (including the artificial category `elPi0`) and is trained on roughly 100000 clusters.

During the training process, the user can see the training (mean squared) error after the given number of iterations. The experiments executed are the following:

1. **Evaluation of the best-trained model on the test dataset** is displayed in the form of a confusion matrix of classified particles. We decided to use this as a "sanity check" to

quickly determine whether the correct data directory was downloaded andwhether it has the expected structure (it is more of a quick check rather than a rigorous check of the whole dataset).

2. **Comparison of the accuracies of the classifier models with different learning parameters by altering their default values displayed in** Table 9.1**:**

validAttributes	TotalEnergy, AverageEnergy, MaxEnergy, PixelCount, Convexity, Width, CrosspointCount, VertexCount, RelativeHaloSize, BranchCount, StdOfEnergy, StdOfArrival and RelLowEnergyPixels
layerSizes	[13, 13]
learningAlgorithm	backProp
epochSize	8
learningRate	0.6
momentum	0.5
activationFunction	sigmoid

Table 9.1 Default model parameters

a) ***Lower learning rate and momentum***
   - learning rate: 0.1, and
   - momentum: 0.1.

b) ***Medium learning rate and momentum***
   - learning rate: 0.5, and
   - momentum: 0.5.

c) ***Higher learning rate and momentum***
   - learning rate: 1, and
   - momentum: 1.

d) ***Small number of hidden layers***
   - A single hidden layer with one neuron.

e) ***Medium number of hidden layers***
   - Two hidden layers with 13 neurons each.

f) ***Higher number of hidden layers***

- Three hidden layers with 13 neurons each.

3. **Comparison of the single-level classifier accuracy with the multi-level classifier.** We trained 10 multi-level and 10 single-level classifiers (we verified the models were properly trained by observing the training error that was not decreasing) and compared their accuracy.

4. **Calculation of the k-fold cross-validation for each type of the single-level classifiers used in the `bestClassifier.csf.`** To check the performance of each single-level classifier, we trained the following types of classifiers:
   a. S_{lead}
   b. $S_{fr\_he\_fe}$
   c. S_{le_pr}
   d. $S_{e\_m\_p\_ep}$

   Then, we calculated their accuracy on a test fold (we performed a 6-fold validation)

Note: the execution of the tests could take a significant amount of time (around an hour for the first two tests each, the last test took less than 10 minutes) because of the large training datasets. The experiments were run on the desktop with the following parameters:

CPU: Intel i5-66000K, 3.5GHz,
GPU: NVIDIA GeForce GTX 1060
RAM: 8GB,
SSD: 256GB,
Address space: 64-bit,
OS: Windows 7.

10 Demonstration

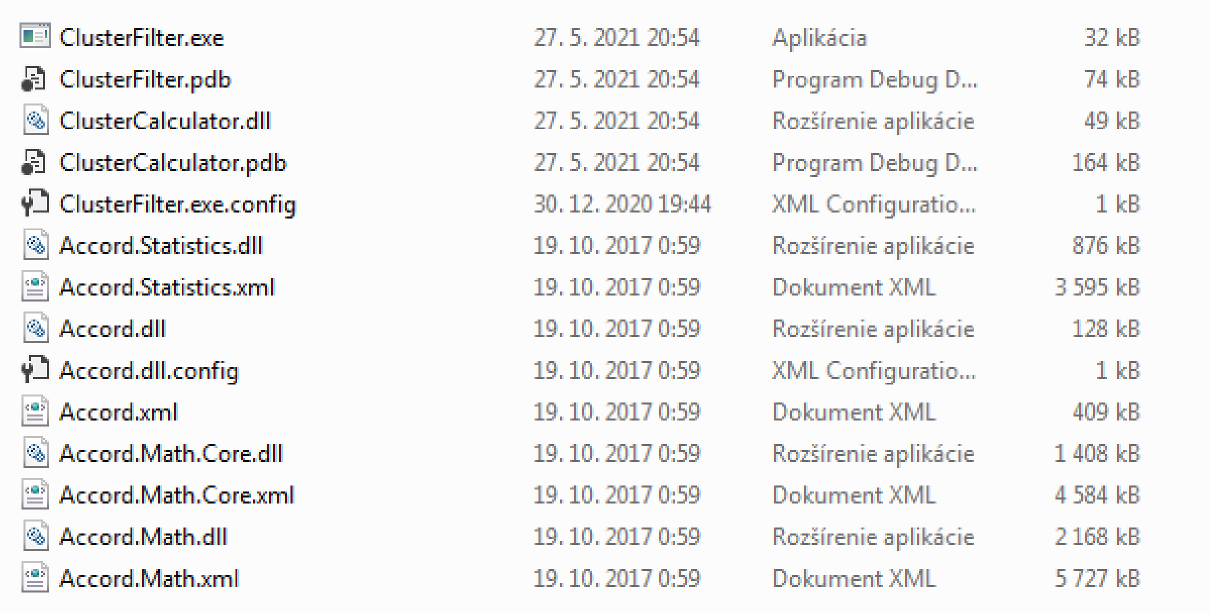
In order to guide the user through the whole ClusterProcessor solution, we decided to create a brief demonstration. Users can go step-by-step and gain intuition on how to use the applications.

10.1 Preparing the data

- Download the binary files or clone the repository at https://github.com/TomSpeedy/ClusterProcessor. For further information, see Chapter 2.
- Download the sample dataset from https://drive.google.com/file/d/1f_t0RE7KiqO69yrzWAbU8LSX8r3l4m_J/view?usp=sharing
- Inspect the downloaded data file – it consists of 4 folders:
  - i. **train_data** are used for training the neural network classifiers in JSON file format. These contain both the examples of clusters used for training as well as training parameters of the networks.
  - ii. **test_data** are data in JSON file format used for testing of the classifiers trained on a different dataset (train_data).
  - iii. **demo_data_MM** represents the clusters in MM format, which are split into multiple directories based on their class (e.g., proton, electron).
  - iv. **trained_models** contain the classifier models which were already trained and can be used for classification or (in the case of single-level classifiers) for further training.

10.2 Filtering

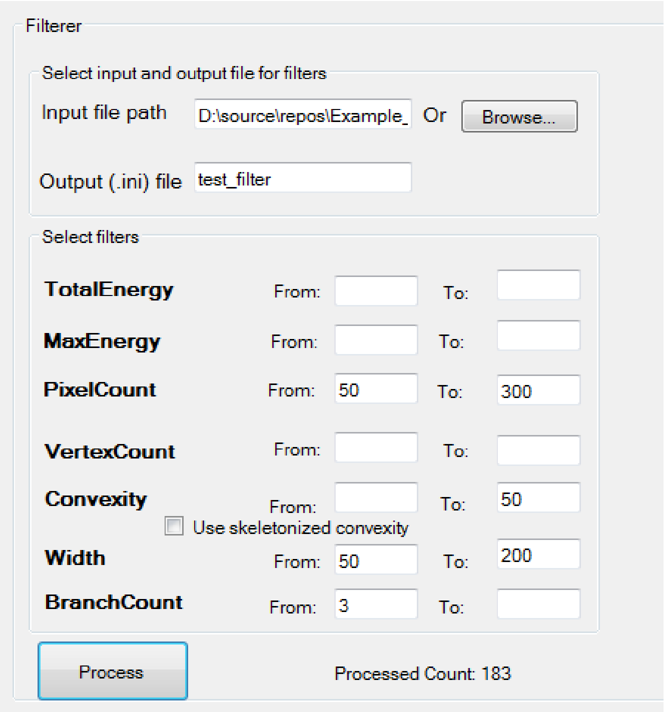
When we have the data and applications prepared, we can navigate to the ClusterFilter binary folder (/bin/Release). Its structure should match the one displayed in Figure 10.1.



ClusterFilter.exe	27. 5. 2021 20:54	Aplikácia	32 kB
ClusterFilter.pdb	27. 5. 2021 20:54	Program Debug D...	74 kB
ClusterCalculator.dll	27. 5. 2021 20:54	Rozšírenie aplikácie	49 kB
ClusterCalculator.pdb	27. 5. 2021 20:54	Program Debug D...	164 kB
ClusterFilter.exe.config	30. 12. 2020 19:44	XML Configuratio...	1 kB
Accord.Statistics.dll	19. 10. 2017 0:59	Rozšírenie aplikácie	876 kB
Accord.Statistics.xml	19. 10. 2017 0:59	Dokument XML	3 595 kB
Accord.dll	19. 10. 2017 0:59	Rozšírenie aplikácie	128 kB
Accord.dll.config	19. 10. 2017 0:59	XML Configuratio...	1 kB
Accord.xml	19. 10. 2017 0:59	Dokument XML	409 kB
Accord.Math.Core.dll	19. 10. 2017 0:59	Rozšírenie aplikácie	1 408 kB
Accord.Math.Core.xml	19. 10. 2017 0:59	Dokument XML	4 584 kB
Accord.Math.dll	19. 10. 2017 0:59	Rozšírenie aplikácie	2 168 kB
Accord.Math.xml	19. 10. 2017 0:59	Dokument XML	5 727 kB

Figure 10.1 Filterer application folder with executable binary

- By double-clicking **ClusterFilter.exe,** the application is run, and the form is opened.
- Then, we click `Browse,` and in the dialog, we select the file **demo_data_MM/frag/k_nuc.ini**
- We also choose the name for our output file as **test_filter.**
- Now, we choose the filtering parameters as displayed in Figure 10.2.
- We click the `Process` button, and we are immediately informed that the filtering is done.
- The output file is in the same directory as the input file and contains three clusters, as displayed in Figure 10.3.



Filterer

Select input and output file for filters

Input file path: D:\source\repos\Example_ Or

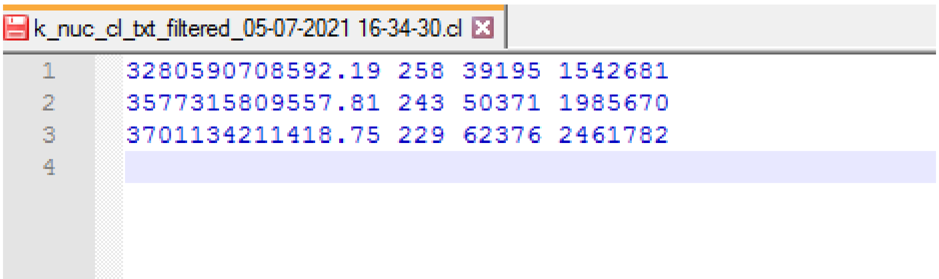
Output (.ini) file: test_filter

Select filters

TotalEnergy	From:		To:	
MaxEnergy	From:		To:	
PixelCount	From:	50	To:	300
VertexCount	From:		To:	
Convexity	From:		To:	50
	□ Use skeletonized convexity			
Width	From:	50	To:	200
BranchCount	From:	3	To:	

Processed Count: 183

Figure 10.2 Filled filterer form

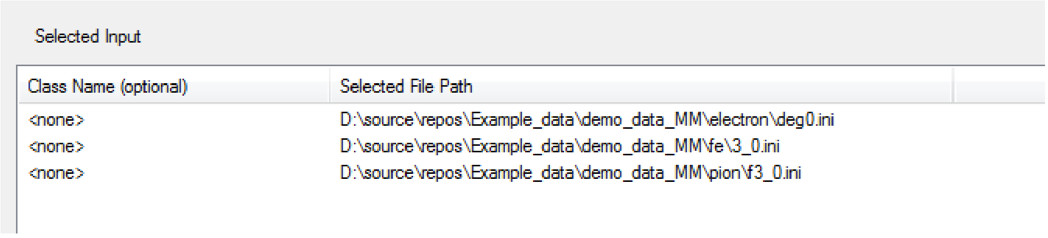


```
3280590708592.19 258 39195 1542681
3577315809557.81 243 50371 1985670
3701134211418.75 229 62376 2461782
```

Figure 10.3 Output file from filterer containing only 3 clusters

10.3 ClusterDescriptionGen

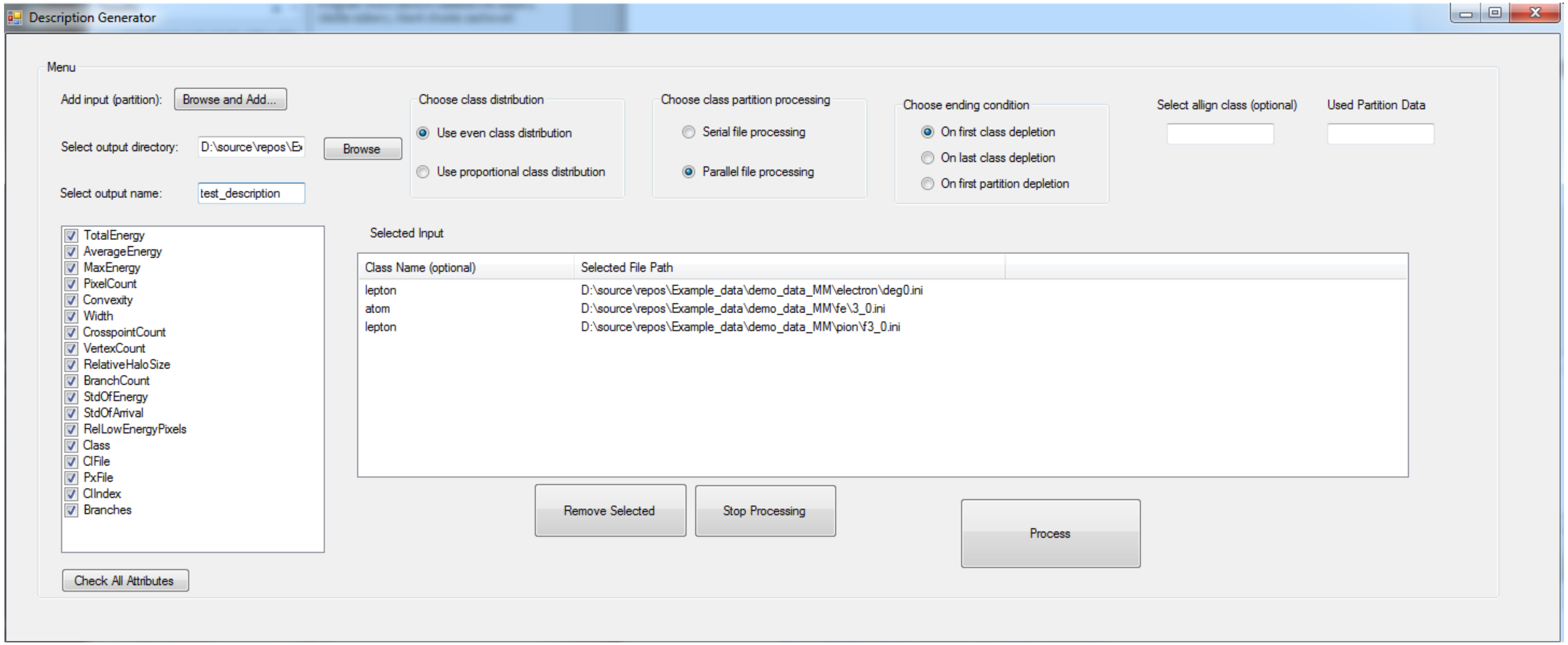
- We navigate to the folder with description generator executable binary (**/bin/Release**).
- By double-clicking on **ClusterDescriptionGen.exe** we open the form.
- We click `Browse and add` button and select **demo_data_MM/electron/deg0.ini**
- We click `Browse and add` button and choose **demo_data_MM/fe/3_0.ini**
- We click `Browse and add` button and choose **demo_data_MM/pion/f3_0.ini**
- The selected partitions are displayed in the form as shown in Figure 10.4.



Class Name (optional)	Selected File Path
<none>	D:\source\repos\Example_data\demo_data_MM\electron\deg0.ini
<none>	D:\source\repos\Example_data\demo_data_MM\fe\3_0.ini
<none>	D:\source\repos\Example_data\demo_data_MM\pion\f3_0.ini

Figure 10.4 Selected inputs (partitions) for description generation

- We can now choose a class name for each of the three partitions – electron and pion are leptons while the iron is an atom, so we can edit the class name based on that by triple-clicking on the class name (which has value `<none>` by default).
- After editing the partitions, we select the name for the input as **test_description** and select the output directory as **train_data** by clicking the `Browse` button
- Then, we can calculate all possible attributes by clicking `Check All Attributes.` The form should look similar to the one displayed in Figure 10.5.
- After clicking `Process`, the program starts processing the input files and creates the output file. It should only take a couple of seconds until we are shown the message box informing us that the processing is over.



Description Generator

Menu

Add input (partition):

Select output directory: D:\source\repos\Ex

Select output name: test_description

Choose class distribution

● Use even class distribution

○ Use proportional class distribution

Choose class partition processing

○ Serial file processing

● Parallel file processing

Choose ending condition

● On first class depletion

○ On last class depletion

○ On first partition depletion

Select allign class (optional)

Used Partition Data

☑ TotalEnergy
☑ AverageEnergy
☑ MaxEnergy
☑ PixelCount
☑ Convexity
☑ Width
☑ CrosspointCount
☑ VertexCount
☑ RelativeHaloSize
☑ BranchCount
☑ StdOfEnergy
☑ StdOfArrival
☑ RelLowEnergyPixels
☑ Class
☑ ClFile
☑ PxFile
☑ ClIndex
☑ Branches

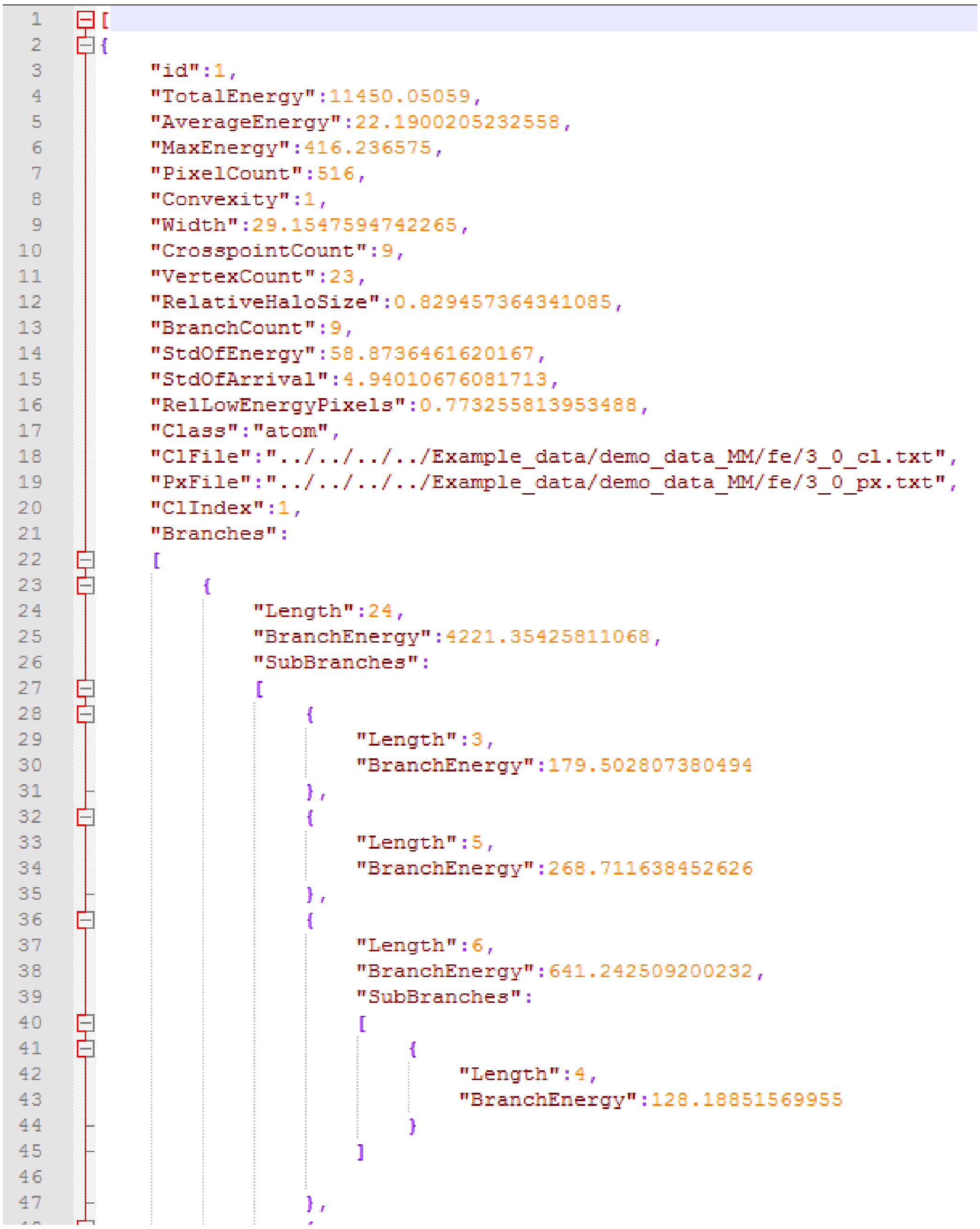
Selected Input

Class Name (optional)	Selected File Path
lepton	D:\source\repos\Example_data\demo_data_MM\electron\deg0.ini
atom	D:\source\repos\Example_data\demo_data_MM\fe\3_0.ini
lepton	D:\source\repos\Example_data\demo_data_MM\pion\f3_0.ini

Figure 10.5 Description generator form before processing

- The output file contains the descriptions starting with a cluster, as shown in Figure 10.6.

  (Note: the `ClFile` and `PxFile` values may be different based on the relative path of the output file to the original files in MM format)



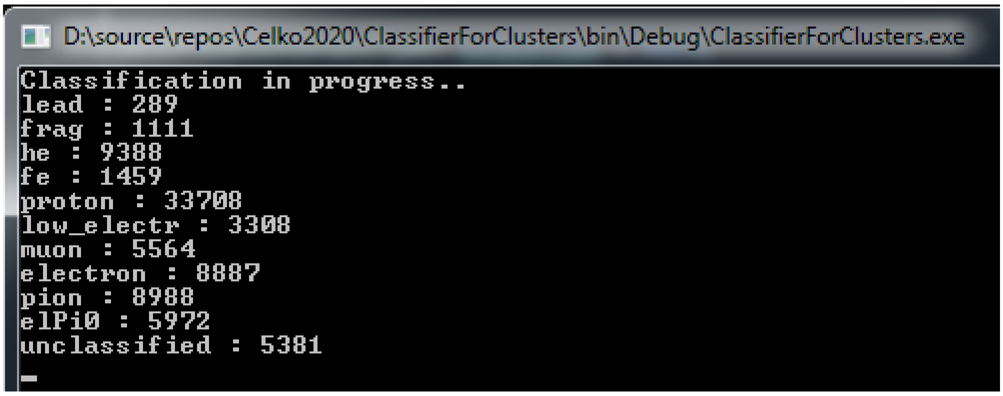
```
[
{
    "id":1,
    "TotalEnergy":11450.05059,
    "AverageEnergy":22.1900205232558,
    "MaxEnergy":416.236575,
    "PixelCount":516,
    "Convexity":1,
    "Width":29.1547594742265,
    "CrosspointCount":9,
    "VertexCount":23,
    "RelativeHaloSize":0.829457364341085,
    "BranchCount":9,
    "StdOfEnergy":58.8736461620167,
    "StdOfArrival":4.94010676081713,
    "RelLowEnergyPixels":0.773255813953488,
    "Class":"atom",
    "ClFile":"../../../../Example_data/demo_data_MM/fe/3_0_cl.txt",
    "PxFile":"../../../../Example_data/demo_data_MM/fe/3_0_px.txt",
    "ClIndex":1,
    "Branches":
    [
        {
            "Length":24,
            "BranchEnergy":4221.35425811068,
            "SubBranches":
            [
                {
                    "Length":3,
                    "BranchEnergy":179.502807380494
                },
                {
                    "Length":5,
                    "BranchEnergy":268.711638452626
                },
                {
                    "Length":6,
                    "BranchEnergy":641.242509200232,
                    "SubBranches":
                    [
                        {
                            "Length":4,
                            "BranchEnergy":128.18851569955
                        }
                    ]

                },
```

Figure 10.6 Beginning of the output of the description generator

10.4 Classifier

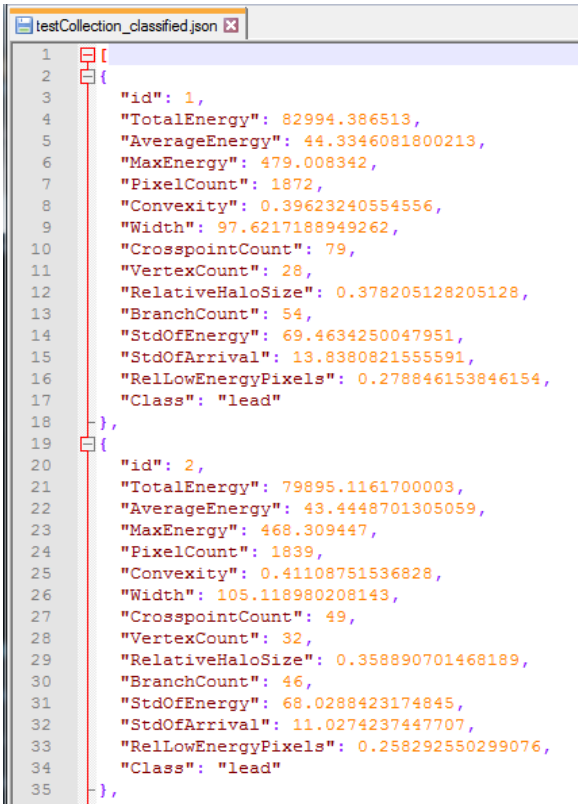
- We navigate to the folder with ClassifierForClusters executable binary (**/bin/Release**).
- We run the executable file **ClassifierForClusters.exe** with the following arguments:
  - [relative or full path to **trained_models/bestClassifier.csf**] (in our case "D:/source/repos/Example_data/trained_models/bestClassifier.csf")
  - [relative or full path to **test_data/testCollection.json**] (in our case, "D:/source/repos/Example_data/test_data/testCollection.json")
  - **--default --singleFile --multi**
- "Classification in progress.." appears, and after a couple of seconds, the particle frequencies are print to the console, see Figure 10.7.


D:\source\repos\Celko2020\ClassifierForClusters\bin\Debug\ClassifierForClusters.exe

```
Classification in progress..
lead : 289
frag : 1111
he : 9388
fe : 1459
proton : 33708
low_electr : 3308
muon : 5564
electron : 8887
pion : 8988
elPi0 : 5972
unclassified : 5381
_
```

Figure 10.7 Console output of the classifier

- When we navigate to the folder test_data, we can see the former collection of clusters (testCollection.json) and the created classified collection file **testCollection_classified.json,** which starts with the clusters displayed in Figure 10.8.



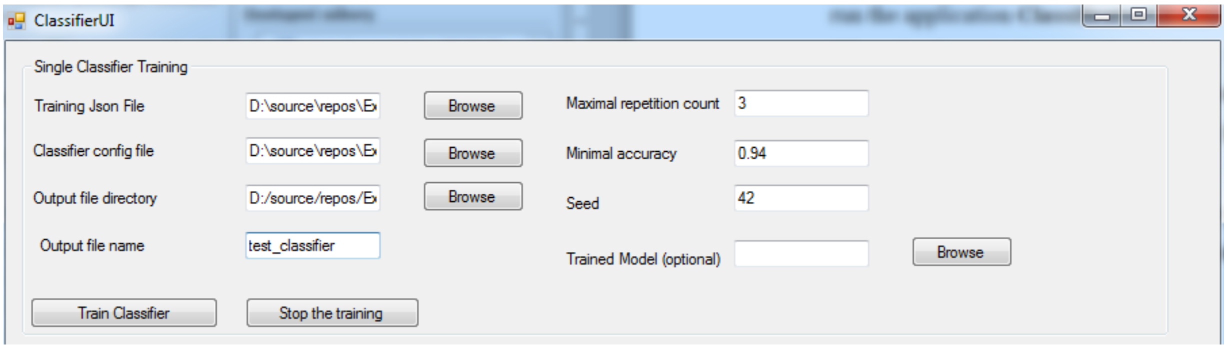
```
[
{
  "id": 1,
  "TotalEnergy": 82994.386513,
  "AverageEnergy": 44.3346081800213,
  "MaxEnergy": 479.008342,
  "PixelCount": 1872,
  "Convexity": 0.39623240554556,
  "Width": 97.6217188949262,
  "CrosspointCount": 79,
  "VertexCount": 28,
  "RelativeHaloSize": 0.378205128205128,
  "BranchCount": 54,
  "StdOfEnergy": 69.4634250047951,
  "StdOfArrival": 13.8380821555591,
  "RelLowEnergyPixels": 0.278846153846154,
  "Class": "lead"
},
{
  "id": 2,
  "TotalEnergy": 79895.1161700003,
  "AverageEnergy": 43.4448701305059,
  "MaxEnergy": 468.309447,
  "PixelCount": 1839,
  "Convexity": 0.41108751536828,
  "Width": 105.118980208143,
  "CrosspointCount": 49,
  "VertexCount": 32,
  "RelativeHaloSize": 0.358890701468189,
  "BranchCount": 46,
  "StdOfEnergy": 68.0288423174845,
  "StdOfArrival": 11.0274237447707,
  "RelLowEnergyPixels": 0.258292550299076,
  "Class": "lead"
},
```

Figure 10.8 Beginning of the labeled clusters file (output of the classifier)

10.5 ClassifierUI

Classifier Training

- We navigate to the folder with ClassificatorUI executable binary (**/bin/Release**) and run the application **ClassificatorUI.exe**.
- When the form (and a console window) is opened, we select the training file by clicking `Browse` and choosing **train_data/trainPrLe_ElMuPi.json.**
- Then, we select the classifier config file by clicking `Browse` and choosing **train_data/PrLeNetworkConfig.json.**
- When we click browse next to the `Output file directory` label, we can select path to **trained_models** directory and then type in the name of classifier file in the textbox right below – **test_classifier**.
- We will leave the trained model field empty because we want to train a new model. We set the Maximal repetition count to 3, minimal accuracy to 0.94, and seed to 42, as displayed in Figure 10.9.



ClassifierUI

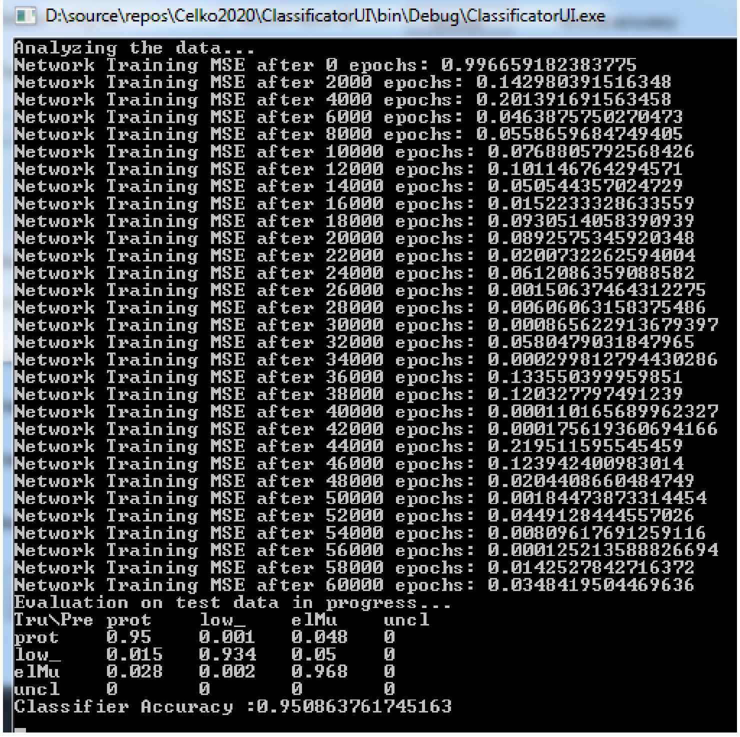
Single Classifier Training

Training Json File	D:\source\repos\E	Maximal repetition count	3
Classifier config file	D:\source\repos\E	Minimal accuracy	0.94
Output file directory	D:/source/repos/E	Seed	42
Output file name	test_classifier	Trained Model (optional)	

Figure 10.9 Filled classifier form before training process

- We click `Train Classifier` button. In the console, "Analyzing the data.. " is displayed, followed by the error rates of the classifier after each iteration, as shown in Figure 10.10.
- The trained classifier files are the two files with the chosen names, where **.csf** and **.csf_support** are added as a sufix.

- Important note: Used implementation (Accord.Net) does not allow network weight initialization based on a seed. The seed is only used for splitting the dataset into training and testing data. This means that user's results might vary slightly when running the program.


D:\source\repos\Celko2020\ClassificatorUI\bin\Debug\ClassificatorUI.exe

```
Analyzing the data...
Network Training MSE after 0 epochs: 0.996659182383775
Network Training MSE after 2000 epochs: 0.142980391516348
Network Training MSE after 4000 epochs: 0.201391691563458
Network Training MSE after 6000 epochs: 0.0463875750270473
Network Training MSE after 8000 epochs: 0.0558659684749405
Network Training MSE after 10000 epochs: 0.0768805792568426
Network Training MSE after 12000 epochs: 0.101146764294571
Network Training MSE after 14000 epochs: 0.050544357024729
Network Training MSE after 16000 epochs: 0.0152233328633559
Network Training MSE after 18000 epochs: 0.0930514058390939
Network Training MSE after 20000 epochs: 0.0892575345920348
Network Training MSE after 22000 epochs: 0.0200732262594004
Network Training MSE after 24000 epochs: 0.0612086359088582
Network Training MSE after 26000 epochs: 0.00150637464312275
Network Training MSE after 28000 epochs: 0.00606063158375486
Network Training MSE after 30000 epochs: 0.000865622913679397
Network Training MSE after 32000 epochs: 0.0580479031847965
Network Training MSE after 34000 epochs: 0.000299812794430286
Network Training MSE after 36000 epochs: 0.133550399959851
Network Training MSE after 38000 epochs: 0.120327797491239
Network Training MSE after 40000 epochs: 0.000110165689962327
Network Training MSE after 42000 epochs: 0.000175619360694166
Network Training MSE after 44000 epochs: 0.219511595545459
Network Training MSE after 46000 epochs: 0.123942400983014
Network Training MSE after 48000 epochs: 0.0204408660484749
Network Training MSE after 50000 epochs: 0.00184473873314454
Network Training MSE after 52000 epochs: 0.0449128444557026
Network Training MSE after 54000 epochs: 0.00809617691259116
Network Training MSE after 56000 epochs: 0.000125213588826694
Network Training MSE after 58000 epochs: 0.0142527842716372
Network Training MSE after 60000 epochs: 0.0348419504469636
Evaluation on test data in progress...
Tru\Pre prot    low_    elMu    uncl
prot    0.95    0.001   0.048   0
low_    0.015   0.934   0.05    0
elMu    0.028   0.002   0.968   0
uncl    0       0       0       0
Classifier Accuracy :0.950863761745163
```

Figure 10.10 Console output during the training process

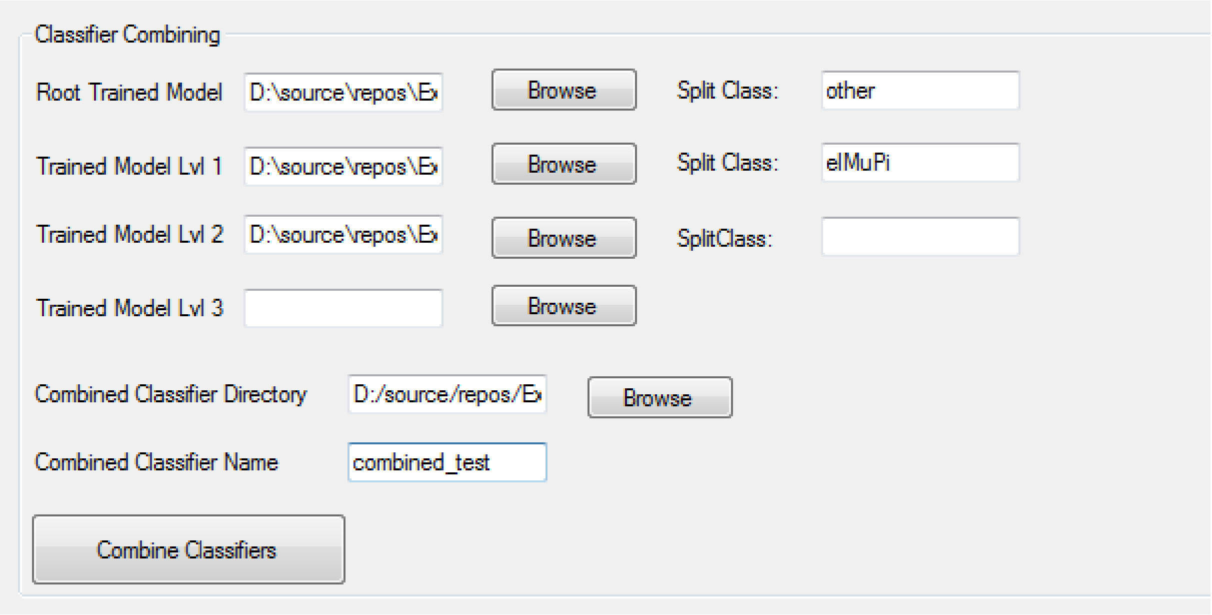
Classifier Cascading

- To combine a couple of trained classifiers into a multi-level classifier, we can use the bottom part of the UI.
- Click `Browse` button next to the `Root Trained Model` label and select **trained_models/trainFragHeFeNew.json_trained_0.961.csf** and type **other** into its `Split Class` field. This classifier splits the data into the following classes:
    - frag,
    - he,
    - fe,
    - and other (electrons, proton, pions, and muons combined).
- Click `Browse` button next to the `Trained Model Lvl 1` label and select **trained_models/trainPrLe_ElMuPi.json_trained_0.966.csf** and type **elMuPi** into its `Split Class` field. This classifier splits the data into the following classes:
    - low energy electron,
    - proton,
    - and other.
- Click `Browse` button next to the `Trained Model Lvl 2` label and select **trained_models/trainElMuPi.json_trained_0.801.csf,** and leave its `Split Class`

field blank as shown in Figure 10.11. This classifier splits the data into the following classes:

  - electron,
  - muon,
  - pion,
  - and elPi0 (electron or pion with an angle of zero degrees).

- We choose the classifier directory **trained_classifiers** and name **combined_test,** and then click the `Combine Classifiers` button.



Classifier Combining

Root Trained Model D:\source\repos\E

Split Class: other

Trained Model Lvl 1 D:\source\repos\E

Split Class: elMuPi

Trained Model Lvl 2 D:\source\repos\E

SplitClass:

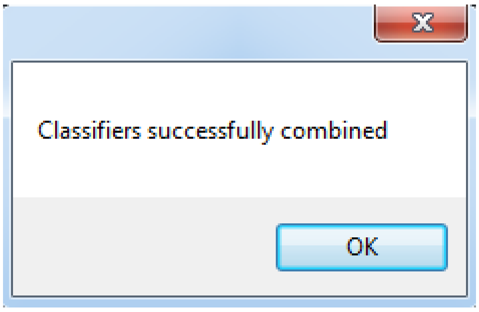
Trained Model Lvl 3

Combined Classifier Directory D:/source/repos/E

Combined Classifier Name combined_test

Figure 10.11 Classifier combining form before the start of combining process

- After that, a user is informed that the combining was successfuly completed, as displayed in Figure 10.12. Then, the combined classifier (in our case) is stored in the selected directory as **combined_test.csf** and **combined_test.csf_support.**



Classifiers successfully combined

Figure 10.12 Message box indicating success of the combining process

10.6 ClassifierTrainer

- We navigate to the folder with ClassifierTrainer executable binary (**/bin/Release**).
- Then, we run the executable file **ClassifierTrainer.exe** with the following parameters:
    - [full or relative path to **train_data/trainPrLe_ElMuPi.json**]
    - [full or relative path to **train_data/PrLeNetworkConfig.json**]
    - [full or relative path to the output file that will be created - we used full path to **trained_models/trained_example**]
    - **--acc 0.94**
    - **--maxrep 3**
    - **--seed 42**
- In the console, "Analyzing the data.. " is displayed, followed by the error rates of the classifier after each iteration and evaluation in the form of a normalized confusion matrix.
- The trained classifier can be found in the chosen directory under the names **trained_example.csf** and **trained_example.csf_support** (the name contains the accuracy of the classifier, which means the actual filename may contain a different number than 0.95)

10.7 ClusterViewer

- We navigate to the folder with ClassifierUI executable binary (**/bin/Release**) and run the application **ClusterUI.exe**.
- Click on the `Browse` button and select **demo_data_MM/lead/deg0.ini.** The collection is loaded, and the following are displayed – information about the current zoom and the name of the viewed `.ini` file, see Figure 10.14.
- Then we fill in the index of a searched cluster **23** and click the `Find` button. A new cluster is loaded with a given index in the collection, as displayed in Figure 10.13.

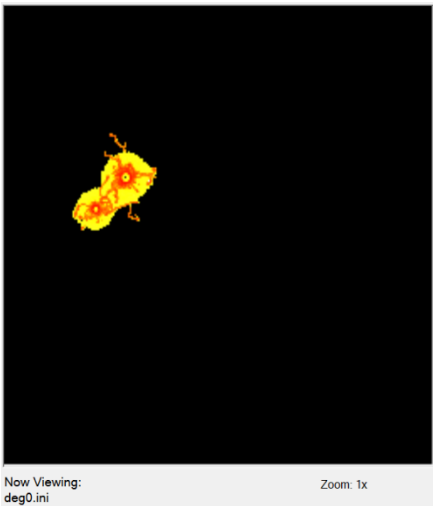


Figure 10.13 Image of the 23rd cluster in the collection

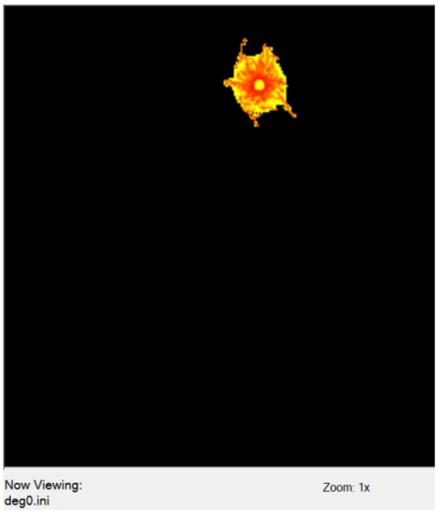


Figure 10.14 Image of the first cluster in the collection

- After clicking `Show Attributes,` the list of attributes of a cluster is displayed, see Figure 10.15.
- When we click `Skeletonize` button, the skeleton of a cluster is shown, as displayed in Figure 10.16.

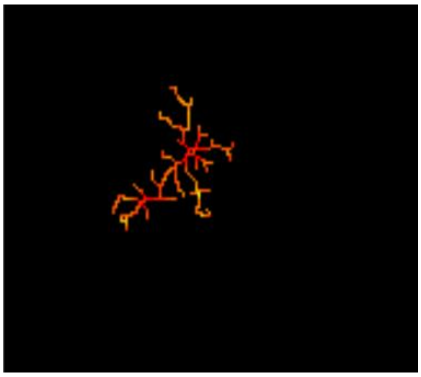
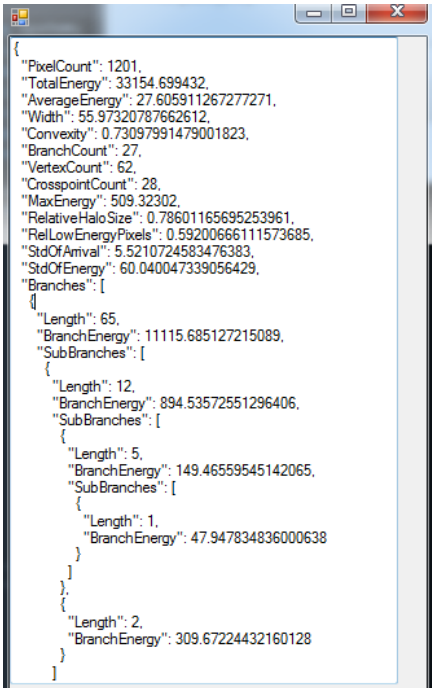


Figure 10.16 Skeletonized cluster



```
{
  "PixelCount": 1201,
  "TotalEnergy": 33154.699432,
  "AverageEnergy": 27.605911267277271,
  "Width": 55.97320787662612,
  "Convexity": 0.73097991479001823,
  "BranchCount": 27,
  "VertexCount": 62,
  "CrosspointCount": 28,
  "MaxEnergy": 509.32302,
  "RelativeHaloSize": 0.78601165695253961,
  "RelLowEnergyPixels": 0.59200666111573685,
  "StdOfArrival": 5.5210724583476383,
  "StdOfEnergy": 60.040047339056429,
  "Branches": [
    {
      "Length": 65,
      "BranchEnergy": 11115.685127215089,
      "SubBranches": [
        {
          "Length": 12,
          "BranchEnergy": 894.53572551296406,
          "SubBranches": [
            {
              "Length": 5,
              "BranchEnergy": 149.46559545142065,
              "SubBranches": [
                {
                  "Length": 1,
                  "BranchEnergy": 47.947834836000638
                }
              ]
            },
            {
              "Length": 2,
              "BranchEnergy": 309.67224432160128
            }
          ]
```

Figure 10.15 Attributes of a cluster in JSON data format

- Then, we go to the cluster with index **35** and click the `Show Branches` button. This displays branches detected in the cluster, as shown in Figure 10.17.
- We click `Browse` again to load the file

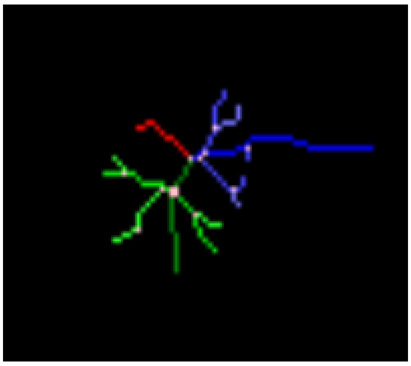


Figure 10.17 Branches found in the cluster

demo_data_MM/frag/k_nuc.ini, search for the 7th cluster and click `Skeletonize` button.

- After that, we click the `View 3D` button and use the rotation buttons to view the trajectory from different angles (see Figure 10.18).
- By clicking `Show Collection Histogram` and `Show Pixel Histogram` buttons, we are displayed the histograms for the whole collection and for the currently viewed cluster, respectively.
- We now click on the `Next` button to move to the next cluster. Then, we click the `Browse` button in the Classification section and choose **trained_models/bestClassifier.csf,** which loads the given classifier into the viewer. Then, we can click on classify for each of the currently viewed clusters, which displays the label with the predicted class (see Figure 10.19).

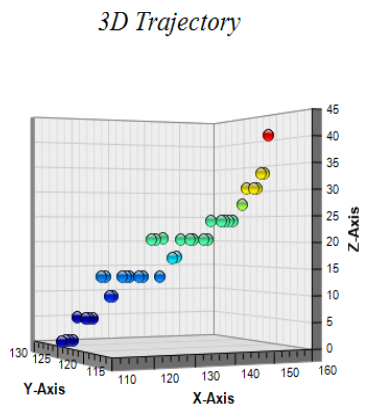
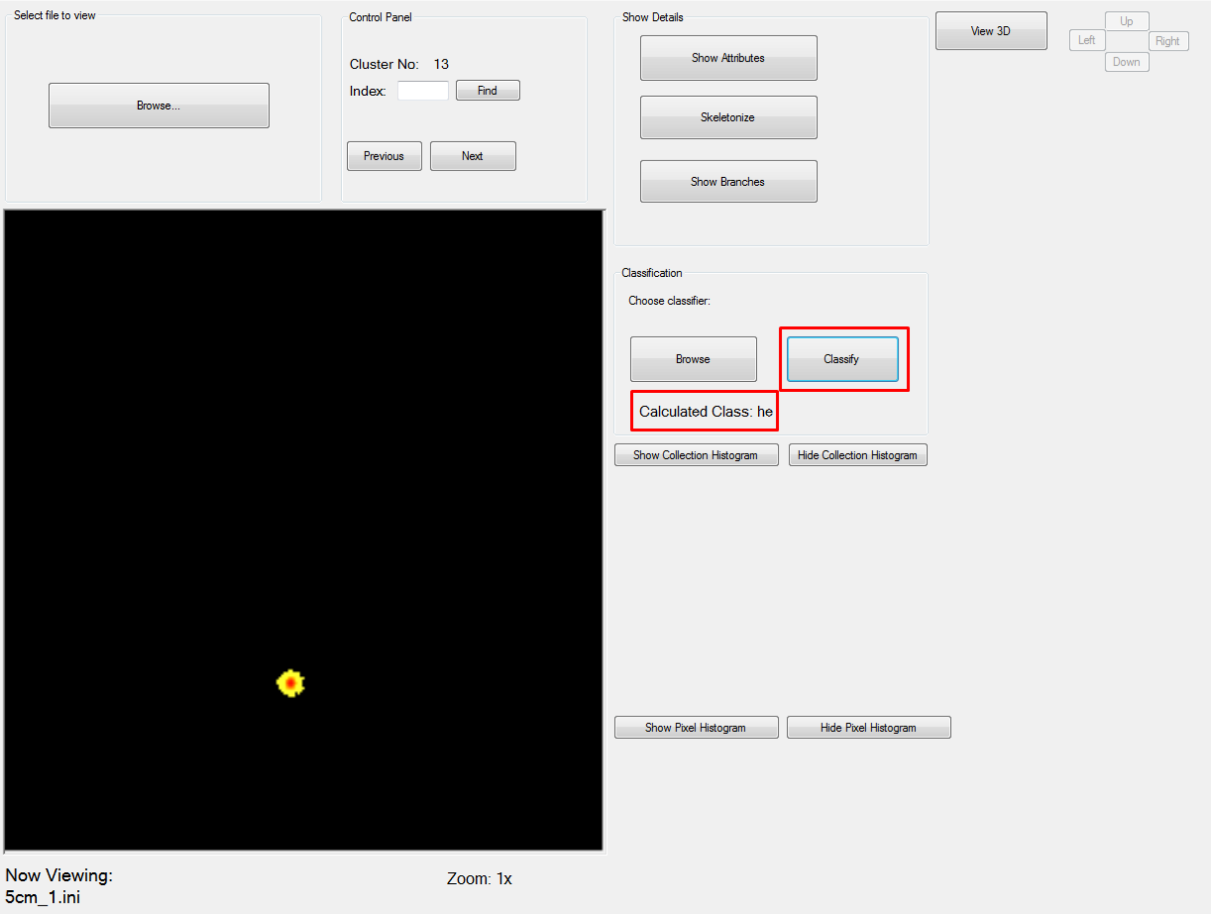


Figure 10.18 3D trajectory of the particle



Cluster No: 13

Index:

Calculated Class: he

Now Viewing:
5cm_1.ini

Zoom: 1x

Figure 10.19 Classification of the current cluster in the cluster viewer

11 Further troubleshooting

In case you have any problems that were not addressed by this manual, or should you have any questions regarding the ClusterProcessor, feel free to contact us at mailto:celko.tom@gmail.com. We appreciate feedback of any kind, and we hope that the application is helpful in anything you would like to use it for.