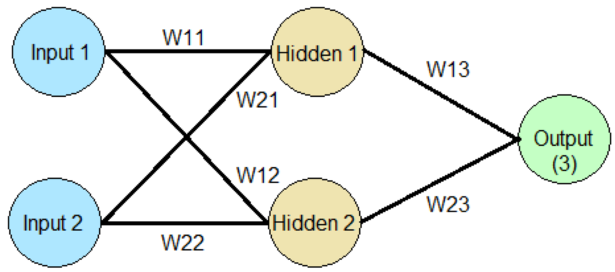
Neural Network MLPClassifier

Release 1.0.7

Aug 08, 2021

Contents


Input 1
Input 2
W11
W21
W12
W22
Hidden 1
Hidden 2
W13
W23
Output
(3)

CHAPTER 1

Context

The XOR problem is a known classification problem, where a two dimensional input space is mapped to a single variable as shown in the table and figure below:

Input 1	Input 2	Output Class
False	False	A
False	True	B
True	False	B
True	True	A

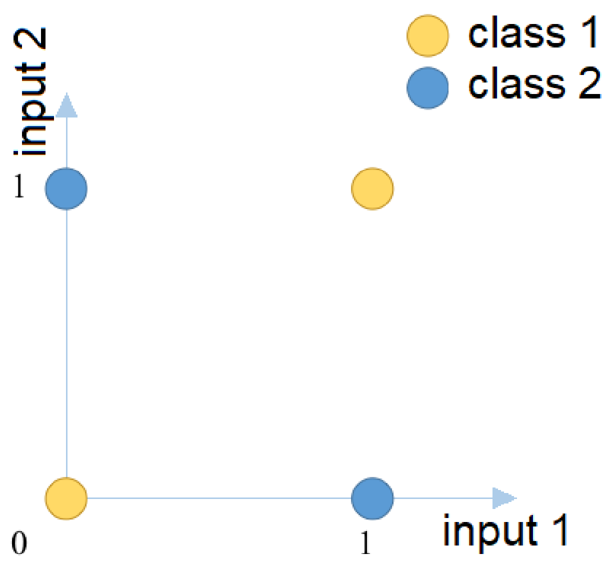


Fig. 1: The XOR classification problem, represented on a graph.

A classification problem like this, where the different classes are no longer linearly separable cannot be solved with e.g. a *minimum distance* or a *maximum likelihood* classifier.

Linearly inseparable classification problems are in practice the rule, rather than the exception. They can be circumvented by splitting up clusters in the input space into multiple sub-clusters before classification, and then merge them again after classification (see table below). However, this is only possible if we have knowledge on the division of the classes in the input space.

Input 1	Input 2	Output Class
False	False	A
False	True	B
True	False	B
True	True	**C**

Artificial Neural Networks are particularly suited to solve this type of linearly inseparable classification problems. The division between the different classes are determined in an iterative process:

(1) Training data is presented to the network.

(2) The network guesses the output based on internal parameters (weights).

(3) This output is compared to the training data and the network error is calculated.

(4) The derivative of the network error to each individual weight is calculated.

(5) The weights are adjusted in the opposite direction of that derivative.

(6) Repeat a fixed number of times, or until the network error falls below a given threshold.

CHAPTER 2

Installation Instructions

For issues, bugs, proposals or remarks, visit the issue tracker.

2.1 Installation of QGIS Plugin

Before installing, you need to install some python packages:

Open the OSGeoShell (comes with the QGIS installation). Make sure you are using the python 3 environment and then install the packages like this:

```
> py3_env
> pip install pyqtgraph
> pip install sklearn
> pip install matplotlib
```

The Neural Network MLP-Classifier is available in the QGIS Python Plugins Repository:

```
Plugins menu > Manage and install plugins... > All > Search for 'Neural Network
↪MLPClassifier'
```

Alternatively, you can download the latest stable distribution (*neuralnetworkmlpclassifier-x-qgis.zip*) and install the plugin manually:

```
Plugins menu > Manage and install plugins... > Install from ZIP > Browse to the zip
↪file > Click *Install plugin*.
```

Note: The plugin is build for QGIS Version 3.6 and up. We recommend to use QGIS version 3.14 or higher. The plugin has been tested on Windows 10.0, Ubuntu 16.04 and Raspbian GNU/Linux 10 (buster).

2.2 Installation of the python package

Open a shell by running the following batch script (adapt to match with your installation). This will open a command prompt with all environmental settings set for use with QGIS:

```
::QGIS installation folder
set OSGEO4W_ROOT=C:\OSGeo4W64

::set defaults, clean path, load OSGeo4W modules (incrementally)
call %OSGEO4W_ROOT%\bin\o4w_env.bat
call qt5_env.bat
call py3_env.bat

::lines taken from python-qgis.bat
set QGIS_PREFIX_PATH=%OSGEO4W_ROOT%\apps\qgis
set PATH=%QGIS_PREFIX_PATH%\bin;%PATH%

::make PyQGIS packages available to Python
set PYTHONPATH=%QGIS_PREFIX_PATH%\python;%PYTHONPATH%

:: GDAL Configuration (https://trac.osgeo.org/gdal/wiki/ConfigOptions)
:: Set VSI cache to be used as buffer, see #6448 and
set GDAL_FILENAME_IS_UTF8=YES
set VSI_CACHE=TRUE
set VSI_CACHE_SIZE=1000000
set QT_PLUGIN_PATH=%QGIS_PREFIX_PATH%\qtplugins;%OSGEO4W_ROOT%\apps\qt5\plugins

::enable/disable QGIS debug messages
set QGIS_DEBUG=1

::open the OSGeo4W Shell
@echo on
@if [%1]==[] (echo run o-help for a list of available commands & cmd.exe /k) else␣
↪(cmd /c "%*")
```

For command-line interface and stand-alone usage, install the python package with pip:

```
pip install mlp-image-classifier
```

For offline installation, you can download the latest stable distribution (*mlp-image-classifier-x.tar.gz*) and:

```
C:\WINDOWS\system32>cd C:\Users\UserName\Downloads
C:\Users\UserName\Downloads>pip install mlp-image-classifier-x.tar.gz
```

CHAPTER 3

User Guide

The Neural Network MLPClassifier can be used in several ways:

1. As a plugin in QGIS
2. From the QGIS processing toolbox
3. As a commandline interface to classify images
4. As a commandline interface to classify *pattern* files
5. Adapting the code to fulfil very specific needs

For the last option, we refer the user to the code repository and the API at the end of this document.

For issues, bugs, proposals or remarks, visit the issue tracker.

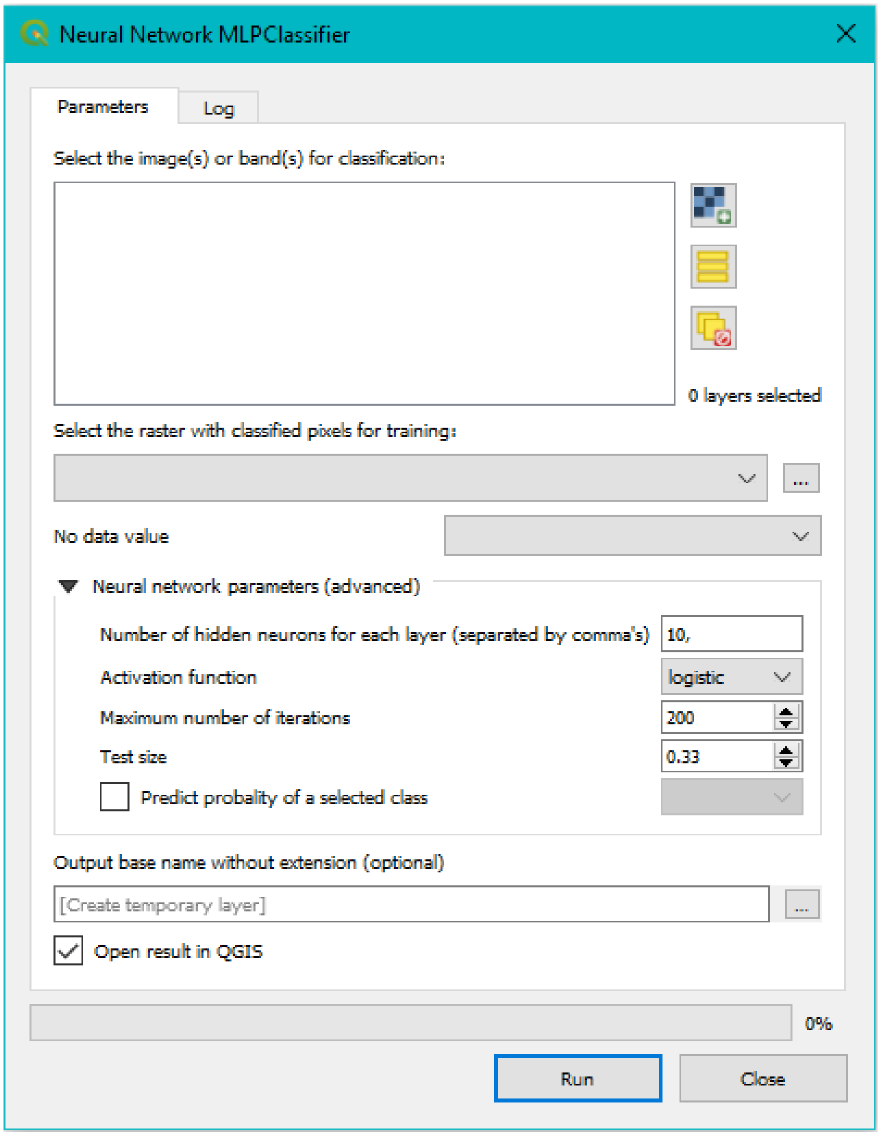
3.1 QGIS Plugin

Required settings:

1. Select the image, or multiple images in case bands are saved in separate images, that you want to classify. At least two bands are required.
2. Select the raster with training pixels. Be aware of value that represents pixels with no training data (*no-data-value*).
3. Set the no-data-value.

Optional settings:

- Choose the number of hidden layers and number of neurons per layer, as a comma separated list.
- Choose another activation function:
  - IDENTITY: no-op activation, useful to implement linear bottleneck, returns f(x) = x;
  - LOGISTIC: logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x));


Neural Network MLPClassifier
Parameters
Log
Select the image(s) or band(s) for classification:
0 layers selected
Select the raster with classified pixels for training:
...
No data value
Neural network parameters (advanced)
Number of hidden neurons for each layer (separated by comma's)
10,
Activation function
logistic
Maximum number of iterations
200
Test size
0.33
Predict probality of a selected class
Output base name without extension (optional)
[Create temporary layer]
...
Open result in QGIS
0%
Run
Close

  - TANH: hyperbolic tan function, returns f(x) = tanh(x);
  - RELU: rectified linear unit function, returns f(x) = max(0, x).
- Choose another number of iterations for training the neural network.
- Set a different test size (the portion of training pixels that will be used to evaluate the trained network).
- Instead of classifying the image, we can predict the probability for each image pixel of finding a given class.
- Choose a filename for the output layer(s).

3.2 Command Line Interface for image classification

The main command is:

```
>mlpclassifier-image
```

Use `-h` or `--help` to list all possible arguments:

```
>mlpclassifier-image -h
```

The **images_folder**, **image_names** and **classes_data_name** are required arguments. Notice how there are **no spaces** between the different band images, in case your file or folder names contain spaces, use double quotes. An example:

```
>mlpclassifier-image folder/to/data band1.tif,band2.tif,band3.tif training_data.tif
```

By default, the classified image is stored in the same folder as the image files, with the name 'output_classified.tif'. To select another file name base (no extension) or another location, use the argument `-o` or `--output`:

```
>mlpclassifier-image folder/to/data band1.tif,band2.tif,band3.tif training_data.tif -
↪o folder/to/output.tif
```

To select a no-data-value for the training data set, use the argument `-n` or `--no_data_value`. The default no-data-value is -1:

```
>mlpclassifier-image folder/to/data band1.tif,band2.tif,band3.tif training_data.tif -
↪n 0
```

To select a different activation function for the MLP classifier (identity, logistic=default, tanh, relu), use the argument `-a` or `--activation`:

```
>mlpclassifier-image folder/to/data band1.tif,band2.tif,band3.tif training_data.tif -
↪a identity
```

To select a different number of iterations (default 200), use the argument `-i` or `--iterations`:

```
>mlpclassifier-image folder/to/data band1.tif,band2.tif,band3.tif training_data.tif -
↪i 2000
```

To select a different number of hidden layers and their neurons, use the argument `-l` or `--hidden_layer_size`:

```
>mlpclassifier-image folder/to/data band1.tif,band2.tif,band3.tif training_data.tif -
↪l 5,5
```

To select a different test size (between 0 and 1) for evaluating the network training, use the argument `-t` or `--test_size`:

```
>mlpclassifier-image folder/to/data band1.tif,band2.tif,band3.tif training_data.tif -
↪t 0.50
```

If, instead of classifying the image, you would like to predict the probability for each image pixel of finding a given class, use the argument `-p` or `--probability_of_class`:

```
>mlpclassifier-image folder/to/data band1.tif,band2.tif,band3.tif training_data.tif -
↪p 4
```

3.3 Command Line Interface for pattern file classification

The main command is:

```
>mlpclassifier-pattern
```

Use `-h` or `--help` to list all possible arguments:

```
>mlpclassifier-pattern -h
```

The **pattern_predict_path** and **pattern_train_path** are required arguments. Use double quotes around paths and file names in case they contain spaces. The default **no-data-value** is -1, change this with the argument `-n` or `--no_data_value` to change. An example:

```
>mlpclassifier-pattern data/folder/predict.prn data/folder/train.prn -n 0
```

The other options are the same as with *mlpclassifier-image*.

CHAPTER 4

Exercises

For issues, bugs, proposals or remarks, visit the issue tracker.

We have developed tree exercises. Two solving the XOR problem (one with the command line, and one with QGIS) and one solving an actual image classification problem.

You can find the data (*test_data.zip*) on the bitbucket page.

4.1 Exercise: XOR problem in QGIS

For issues, bugs, proposals or remarks, visit the issue tracker.

4.1.1 Tutorial Data Set

We will use two images:

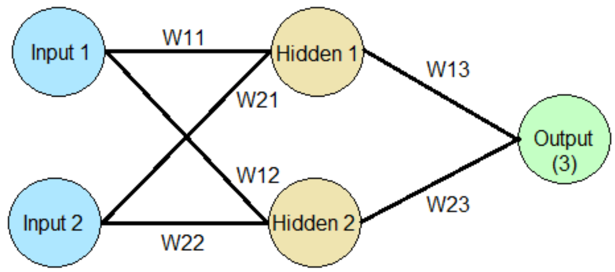
- *xor_training.tiff* for training and
- *xor.tiff* for predicting.

4.1.2 Goal

We will solve the XOR problem (see context) with the MLP Classifier.

In order to do this, we will need a Neural Network with 3 layers: an input layer with 2 input neurons, a hidden layer with 2 neurons and an output layer with one neuron.

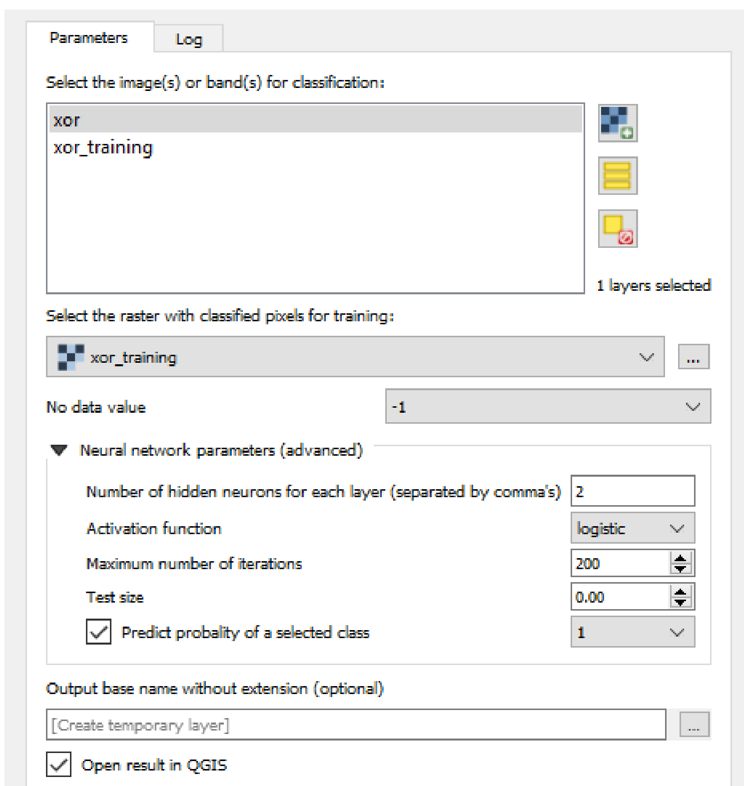
We will however not do a normal classification. Instead, we will calculate the probability for each input that the output is class 1! Have another look at the training data, we have two classes: class 1 and class 2.



4.1.3 Run the Neural Network MLPClassifier in QGIS

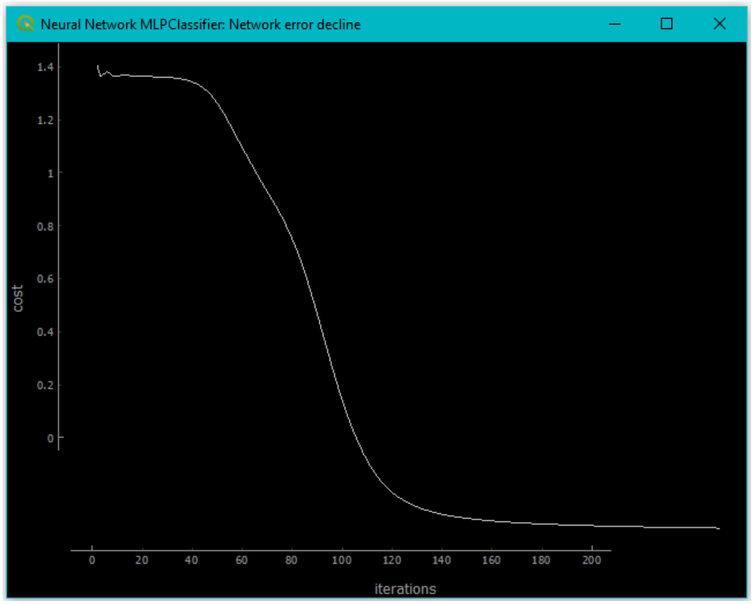
We need the following input for the GUI:

- The image for classification: xor
- The raster for training: xor_training
- The no-data-value is -1
- Number of neurons for each hidden layer: 2 neurons in 1 hidden layer
- Test size: 0
- We want the probability of class 1 (we have two classes, 1 and 2)
- Output file path



After the training, the network error is shown as a plot like:

The resulting image should look something like this:



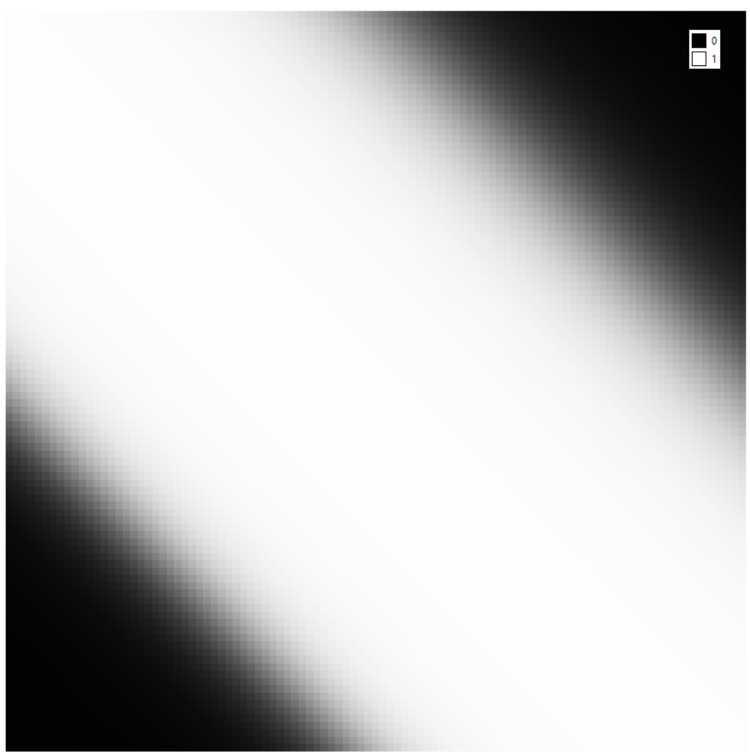


Fig. 1: Probability of each pixel being class 1, ranging from 0 (black, zero probability) to 1 (white, max certainty)

In theory there are two decision surfaces, the image above only shows one. Can you explain?

4.2 Exercise: XOR problem using a *pattern* file and the CLI

For issues, bugs, proposals or remarks, visit the issue tracker.

4.2.1 Tutorial Data Set

We will use a special file format, the *pattern* file (.prn), which is a text file of the following form:

```
number_of_patterns: 4
number_of_inputs: 2
number_of_outputs: 1

0 0 2
0 1 1
1 0 1
1 1 2
```

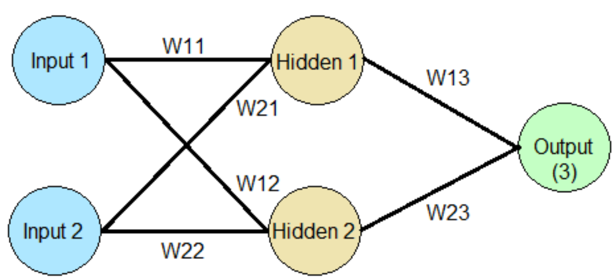
We need two files, one for training, and one for predicting:

- *xor.prn* for training,
- *xor_grid.prn* for predicting.

4.2.2 Goal

We will solve the XOR problem with the MLP Classifier. We can use special pattern files (.prn) for this purpose, however this is only possible in the CLI, as these are text files and not images.

In order to do this, we will need a Neural Network with 3 layers: an input layer with 2 input neurons, a hidden layer with 2 neurons and an output layer with one neuron.



We will however not do a normal classification. Instead, we will calculate the probability for each input that the output is class 1! Have another look at the training data, we have two classes: class 1 and class 2.

4.2.3 Run the CLI

We need the following input for the CLI:

- the path to the file we want to predict
- the path to the training file

- [-n] the no-data-value is -1 (default)
- [-l] the number of neurons per hidden layer –> 2 neurons in 1 hidden layer
- [-t] the test size is 0
- [-p] the class we want the probability for: class 1
- [-o] the output file path (optional)

We keep the default values for the activation function (logistic) and the number of iterations (200).

The resulting command should look at the least something like:

```
mlpclassifier-pattern "C:/.../xor_grid.prn" "C:/.../xor.prn" -l 2 -t 0 -p 1
```

You will have two output files: the prediction of the patterns and a graph of the error decline.

You can plot the results using R, Excel, matplotlib, pyqtgraph, ... See the QGIS exercise for a discussion of the results.

4.3 Exercise: Classify an image in QGIS

For issues, bugs, proposals or remarks, visit the issue tracker.

4.3.1 Tutorial Data Set

We will use several images:

- WINDOWED_SPOT_XI_VIETNAM_011.tif, etc.. for predicting
- reference.tif for training.

We have several tif images at our disposal: band 11, 12, 13 and 14 all saved as separate bands, we have an image with band 11 and 12, and an image with all 4 bands.

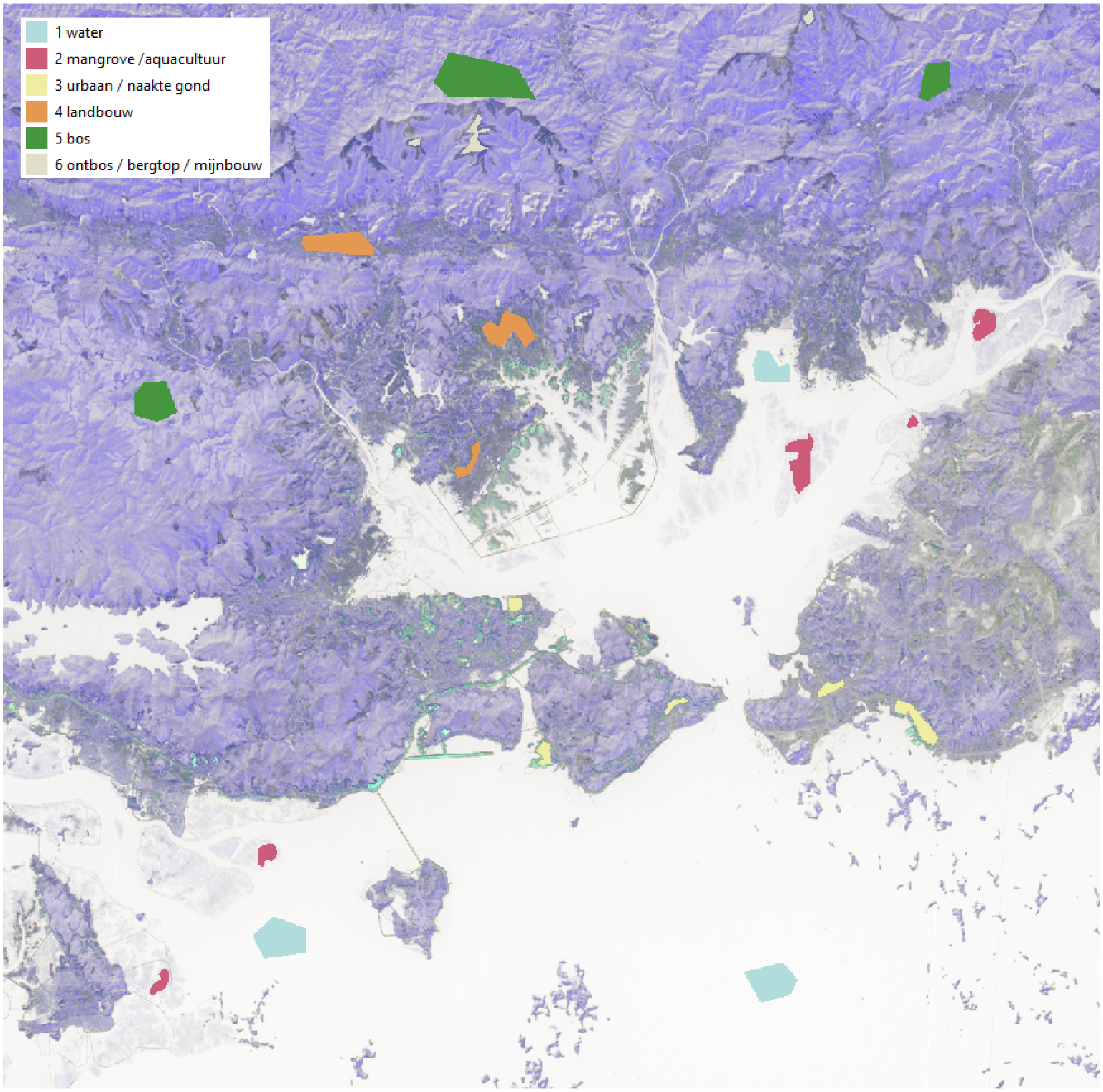
The classification in the reference.tif file is the following:

- 1: Water
- 2: Mangrove/aquaculture
- 3: Urban/bare white ground
- 4: Agriculture
- 5: Forest
- 6: Mountaintops/mining
- 0: No data

4.3.2 Goal

We will do a supervised classification MLP Classifier.

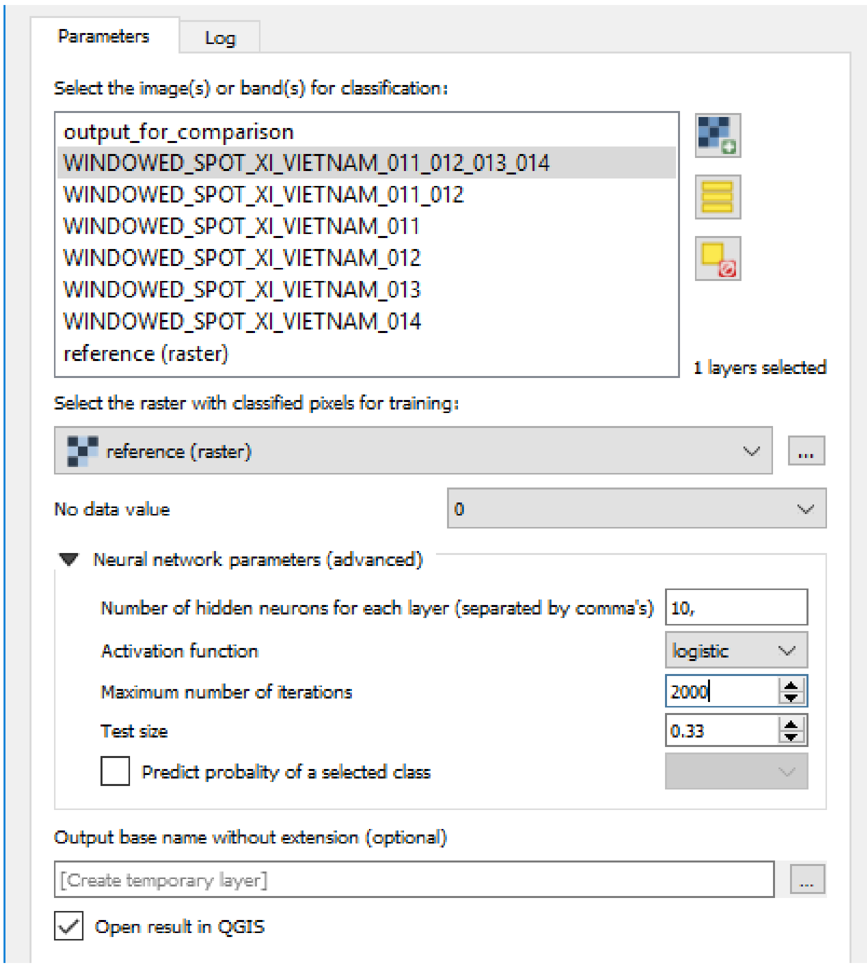
Hint: Use the style file in the exercise folder (*classification_style_raster.qml*) for easy styling.


1 water
2 mangrove /aquacultuur
3 urbaan / naakte gond
4 landbouw
5 bos
6 ontbos / bergtop / mijnbouw

4.3.3 Run the Neural Network MLPClassifier in QGIS

We need the following input for the GUI:

- The image for classification: make sure you select either the image with 4 bands, or the 4 separate bands, … but do not select them twice!
- The raster for training: reference.tif
- Number of iterations: 2000
- No-data-value: 0 (do not forget to set this one!)



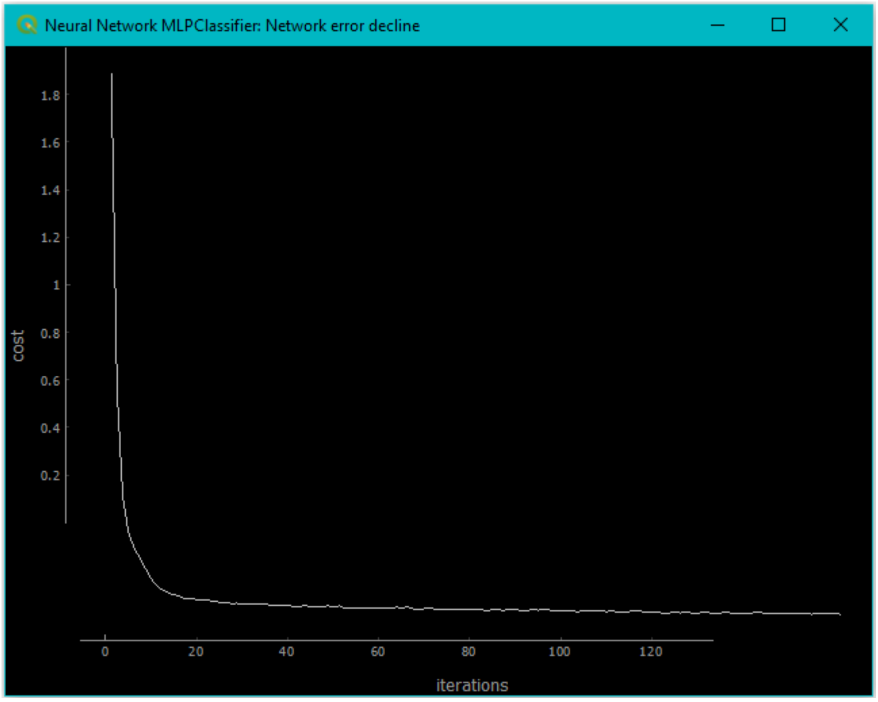
After the training, the network error is shown as a plot like:

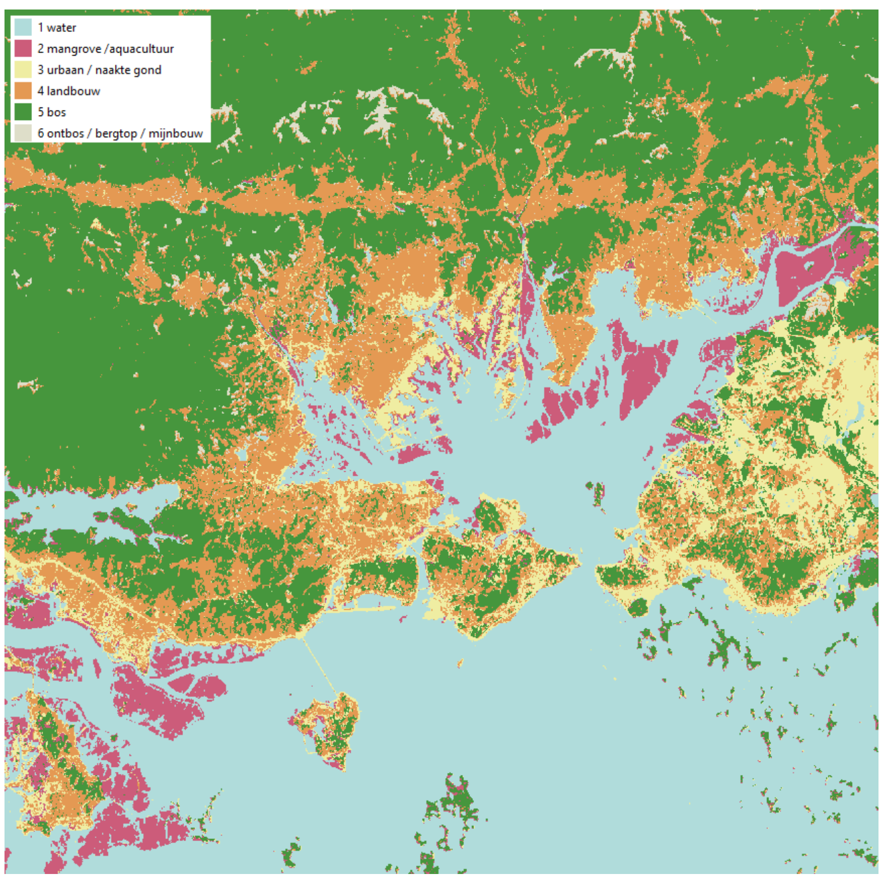
The resulting image should look something like this:

You will also get an output on the log screen with the classification kappa

```
Average accuracy: 0.9804091040046097
Kappa class 1: 0.999608231804873
Kappa class 2: 0.998816901840517
Kappa class 3: 0.943025695066823
Kappa class 4: 0.9422660833150642
Kappa class 5: 0.9909993784914707
Kappa class 6: 0.8952379269755105
```

(continues on next page)


Neural Network MLPClassifier: Network error decline
cost
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
20
40
60
80
100
120
iterations


1 water
2 mangrove /aquacultuur
3 urbaan / naakte gond
4 landbouw
5 bos
6 ontbos / bergtop / mijnbouw

(continued from previous page)

```
Average Kappa: 0.961659036249043

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1685
           1       1.00      1.00      1.00       985
           2       0.96      0.93      0.95       584
           3       0.93      0.98      0.95      1121
           4       0.99      0.99      0.99      2351
           5       0.97      0.84      0.90       216

   micro avg       0.98      0.98      0.98      6942
   macro avg       0.98      0.96      0.97      6942
weighted avg       0.98      0.98      0.98      6942
 samples avg       0.98      0.98      0.98      6942
```

CHAPTER 5

Neural Network MLPClassifier API

Source code: https://bitbucket.org/kul-reseco/lnns/src.

For issues, bugs, proposals or remarks, visit the issue tracker.

5.1 Algorithms

class `lnns.core.network.`**`Network`**(*number_of_hidden*, *activation*)

Bases: `object`

The Network class create a neural network using the sklearn.neural_network.MLPClassifier. The network can be used to predict classified images using supervised classification.

`predict_image`(*band_data*, *probability_of_class=None*)

Use the trained network, it is possible to classify the complete image using the different bands of the image. If probability_of_class is given, than the image will show the probability that this class will occur.

Parameters

- **`band_data`** (*`np.array`*) – array with all bands
- **`probability_of_class`** (*`int`*) – class for which you would like the probability image

Returns np.array classified_image: the classified image

`train_image`(*band_data*, *classes_data*, *max_iter*, *no_data_value=0*, *test_size=0.33*, *log_function=<built-in function print>*)

The given network can be trained given an image of different band waves (band_data) and a respectively data set indicating a subset of different classes of the image (class_data).

Parameters

- **`band_data`** (*`np.array`*) – array with all bands
- **`classes_data`** (*`np.array`*) – an overlap image indicating different classes
- **`max_iter`** (*`int`*) – the number of iterations when training the neural network

- **no_data_value** (*int*) – value that describes pixels with no data in the classes_data file
- **test_size** (*float*) – the proportion of the test_size that will be used to evaluate the trained network
- **log_function** – function to log

validate (*x_test*, *y_test*, *log_function=<built-in function print>*)

The trained neural network can be evaluated using a test set of the data. The validation results will be saved in the validation_results dictionary with the keys: Average accuracy, a Kappa for each predicted class, an average kappa and a rapport including precision, recall, f1-score and support.

Example output:

Average accuracy: 0.9740708729472775
Kappa class 1: 0.9996048676750681
Kappa class 2: 0.9969686283161031
Kappa class 3: 0.9269671354418852
Kappa class 4: 0.9384942730689072
Kappa class 5: 0.9854574554108237
Kappa class 6: 0.8991142021469177
Average Kappa: 0.9577677603432843

.	precision	recall	f1-score	support
0	1.00	1.00	1.00	1664
1	1.00	1.00	1.00	956
2	0.91	0.96	0.93	597
3	0.96	0.94	0.95	1149
4	1.00	0.98	0.99	2345
5	0.88	0.92	0.90	231
avg / total	0.98	0.98	0.98	6942

Parameters

- **x_test** (*np.array[float]*) – test input variables
- **y_test** (*np.array[float]*) – test output values
- **log_function** – function to log

5.2 Interfaces

Date : August 2018

Email : acrabbe.foss@gmail.com
Acknowledgements : Translated from LNNS 1.0 A neural network simulator [C++ software]
Ghent University, Laboratory of Forest Management and Spatial Information Techniques
Lieven P.C. Verbeke

`lnns.interfaces.`**`plot`**(*x, y, size=None, title='Neural Network MLPClassifier: Network error decline', testing=False*)

`lnns.interfaces.`**`plot_to_file`**(*x, y, path, title='Neural Network MLPClassifier: Network error decline'*)

`lnns.interfaces.`**`run_algorithm_images`**(*image_paths, classified_path, no_data_value, hidden_layer_size, activation, iterations, test_size, probability_of_class, update_process_bar=<built-in function print>, log_function=<built-in function print>, output_path=None, plot_path=None, feedback=None, context=None*)

Process all input, in order to run the nn script.

Parameters

- **`image_paths`** – the absolute path to the input raster files
- **`classified_path`** – absolute path to the classified raster file
- **`no_data_value`** – value that describes pixels with no data in the classes_data file
- **`hidden_layer_size`** – Hidden layer size with length = n_layers - 2, comma separated values.
- **`activation`** – Activation function for the MLPClassifier, choices=['identity', 'logistic', 'tanh', 'relu']
- **`iterations`** – Maximum number of iterations
- **`test_size`** – the proportion of the test_size that will be used to evaluate the trained network
- **`probability_of_class`** – class for which you would like the probability image
- **`update_process_bar`** – function to update the progress bar
- **`log_function`** – function to log
- **`output_path`** – path for output files (optional)
- **`plot_path`** – necessary for the processing tool: path for the plot
- **`feedback`** – necessary for the processing tool
- **`context`** – necessary for the processing tool

Returns output path

`lnns.interfaces.`**`run_algorithm_pattern`**(*pattern_train*, *pattern_predict*, *no_data_value*, *hidden_layer_size*, *activation*, *iterations*, *test_size*, *probability_of_class*, *log_function=<built-in function print>*, *output_path=None*, *plot_path=None*, *feedback=None*, *context=None*)

Process all input, in order to run the nn script.

Parameters

- **`pattern_train`** – the absolute path to the pattern file with training data
- **`pattern_predict`** – absolute path to the pattern file for predictions
- **`no_data_value`** – value that describes pixels with no data in the classes_data file
- **`hidden_layer_size`** – Hidden layer size with length = n_layers - 2, comma separated values.
- **`activation`** – Activation function for the MLPClassifier, choices=['identity', 'logistic', 'tanh', 'relu']
- **`iterations`** – Maximum number of iterations
- **`test_size`** – the proportion of the test_size that will be used to evaluate the trained network
- **`probability_of_class`** – class for which you would like the probability image
- **`log_function`** – function to log
- **`output_path`** – path for output files (optional)
- **`plot_path`** – necessary for the processing tool: path for the plot
- **`feedback`** – necessary for the processing tool
- **`context`** – necessary for the processing tool

Returns output path

`lnns.interfaces.`**`tuple_int`**(*s*)

`lnns.interfaces.`**`tuple_int_cli`**(*s*)

`lnns.interfaces.`**`tuple_string`**(*s*)

class `lnns.interfaces.imports.`**`PatternFile`**(*number_of_patterns*, *input_neurons*, *output_neurons*, *x*, *y*)

Bases: `object`

`lnns.interfaces.imports.`**`check_path`**(*path*)

Check if path exists. Skipp path which are in memory

Parameters **`path`** – the absolute path to the input file

`lnns.interfaces.imports.`**`import_image`**(*path*, *reflectance=False*)

Browse for an image.

Parameters

- **`path`** – the absolute path to the image
- **`reflectance`** – return reflectance values instead of DN between 0 and 254

Returns float32 numpy array [#good bands x #rows x #columns] `lnns.interfaces.imports.`**`read_pattern`**(*path_prn*)

`lnns.interfaces.exports.`**`write_classified_image`**(*file_path*, *image*, *geo_transform=None*, *projection=None*, *gdal_type=<Mock name='mock.gdal.GDT_Int16' id='140574537438160'>*)

`lnns.interfaces.exports.`**`write_pattern`**(*output_file*, *number_of_patterns*, *input_neurons*, *output_neurons*, *array*, *fmt*)

5.3 CLI: mlpclassifier-image

The Network class create a neural network using the sklearn.neural_network.MLPClassifier. The network can be used to predict classified images using supervised classification.

```
usage: mlpclassifier-image [-h] [-n NO_DATA_VALUE] [-l HIDDEN_LAYER_SIZE]
                           [-a {identity,logistic,tanh,relu}] [-i ITERATIONS]
                           [-t TEST_SIZE] [-p PROBABILITY_OF_CLASS]
                           [-o OUTPUT] [-g OUTPUT_GRAPH]
                           images_folder image_names classes_data_name
```

5.3.1 Positional Arguments

images_folder Path to the input image data and the classes data.

image_names Name(s) of different image bands.

classes_data_name Name of an overlap image indicating different classes

5.3.2 Named Arguments

-n, --no_data_value Value that describes pixels with no data in the classes_data file (default: -1).

Default: -1

-l, --hidden_layer_size Hidden layer size with length = n_layers - 2, comma separated values. The ith element represents the number of neurons in the ith hidden layer. (default: 10,)

Default: (10,)

-a, --activation Possible choices: identity, logistic, tanh, relu

Activation function for the MLPClassifier (default: logistic).

Default: "logistic"

-i, --iterations Maximum number of iterations (default: 200).

Default: 200

-t, --test_size Portion of test pixels used to evaluate the trained network (default: 0.33).

Default: 0.33

-p, --probability_of_class Class for which you would like the probability image (default: None).

-o, --output Output predicted file (default: in same folder with name 'output_classified.tif'

-g, --output_graph Output error graph (default: in same folder with name 'output_error.PNG'

5.4 CLI: mlpclassifier-pattern

The Network class create a neural network using the sklearn.neural_network.MLPClassifier. The network can be used to predict classified images using supervised classification.

```
usage: mlpclassifier-pattern [-h] [-n NO_DATA_VALUE] [-l HIDDEN_LAYER_SIZE]
                             [-a {identity,logistic,tanh,relu}]
                             [-i ITERATIONS] [-t TEST_SIZE]
                             [-p PROBABILITY_OF_CLASS] [-o OUTPUT]
                             [-g OUTPUT_GRAPH]
                             pattern_predict_path pattern_train_path
```

5.4.1 Positional Arguments

pattern_predict_path Pattern text file that includes the values to predict,'number_of_patterns', 'number_of_inputs' and 'number_of_outputs.'

pattern_train_path Pattern text file that includes the network training values, 'number_of_patterns', 'number_of_inputs' and 'number_of_outputs.'

5.4.2 Named Arguments

-n, --no_data_value Value that describes pixels with no data in the classes_data file (default: -1).

Default: -1

-l, --hidden_layer_size Hidden layer size with length = n_layers - 2, comma separated values. The ith element represents the number of neurons in the ith hidden layer. (default: 10,)

Default: (10,)

-a, --activation Possible choices: identity, logistic, tanh, relu

Activation function for the MLPClassifier (default: logistic).

Default: "logistic"

-i, --iterations Maximum number of iterations (default: 200).

Default: 200

-t, --test_size Portion of test pixels used to evaluate the trained network (default: 0.33).

Default: 0.33

-p, --probability_of_class class for which you would like the probability image (default: None).

-o, --output Output predicted file (default: in same folder with extension '_predict.prn'

-g, --output_graph Output error graph (default: in same folder with extension '_error.PNG'

CHAPTER 6

About the Neural Network MLPClassifier

The *Neural Network MLPClassifier* software package is both a QGIS plugin and stand-alone python package that provides a supervised classification method for multi-band passive optical remote sensing data. It uses an MLP (Multi-Layer Perception) Neural Network Classifier and is based on the Neural Network MLPClassifier by scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

The program was originally developed by Lieven P.C. Verbeke (Ghent University, Laboratory of Forest Management and Spatial Information Techniques). It was written in C++ and ported to PyQGIS in 2019 - 2020. It has been developed in an open source environment to encourage further development of the tool.

The **instruction pages** can be found at <https://mlp-image-classifier.readthedocs.io>.

The code **repository** can be found at https://bitbucket.org/kul-reseco/lnns.

PLEASE GIVE US CREDIT

When using the Neural Network MLPClassifier, please use the following citation:

Crabbé, A. H., Cahy, T., Somers, B., Verbeke, L.P., Van Coillie, F. (2020). Neural Network MLPClassifier (Version x.x) [Software]. Available from https://bitbucket.org/kul-reseco/lnns.

ACKNOWLEDGEMENTS

The software and user guide are based on the Linear Neural Network Simulator [C++ software]: Ghent University, Laboratory of Forest Management and Spatial Information Techniques, Lieven P.C. Verbeke

The revision into a QGIS plugin is funded primarily through BELSPO (the Belgian Science Policy Office) in the framework of the STEREO III Programme – Project LUMOS - SR/01/321.

The LUMOS logo was created for free at https://logomakr.com.

SOFTWARE LICENSE

6.1 Indexes

- genindex
- modindex

Python Module Index

l

Index