GeoDict-AI

Training Neural Networks

User Guide

GeoDict release 2023

Published: May 11, 2023

GeoDict

https://doi.org/10.30423/userguide.geodict

Citation:
Janine Hilden, Rolf Westerteiger, Olga Lykhachova, Barbara Planas. GeoDict 2023 User Guide. GeoDict-AI handbook. Math2Market GmbH, Germany, doi.org/10.30423/userguide.geodict

The handbooks in the User Guide series of Math2Market GmbH can be obtained from:

Math2Market GmbH
Richard-Wagner-Strasse 1
67655 Kaiserslautern
Germany

Phone: +49 631 205 605 0
Fax: +49 631 205 605 99
Email: info@math2market.de
Web: www.math2market.de

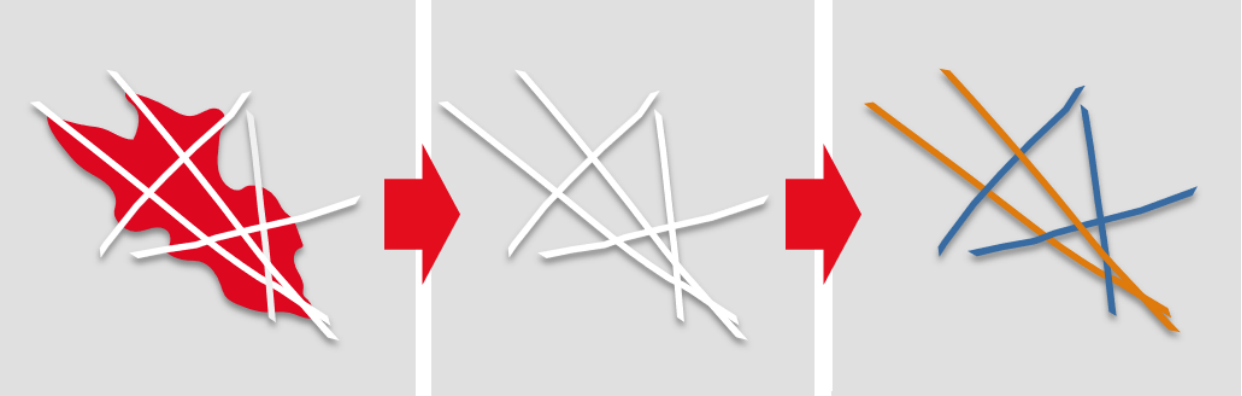
TRAIN NEURAL NETWORKS WITH GEODICT-AI

INTRODUCTION

The GeoDict-AI module covers the functionalities in GeoDict that aim to reconstruct 3D models obtained from segmented computer tomography or FIB/SEM images of different multi-component materials, e.g. nonwovens and electrodes.

The GeoDict-AI module provides the possibility of using **Artificial Intelligence** (AI) approaches for the separation of the different material while image segmentation.

Often, different materials as for example binder and fibers have the same gray values in the images. Therefore, an automatic separation based on the gray value is not possible. However, it is possible to separate the different materials based on their shape in the image. A neural network can be trained to identify binder in a grain structure, to differentiate two fiber types with different shapes or even to identify the individual fibers. In the graphic it is shown how binder is identified from fibers and fibers from each other.



GeoDict-AI is the starting point to analyze physical properties on the segmented 3D-scans considering the different materials. GeoDict-AI can be used for nearly every structure that can be generated with the GeoDict structure generator modules, e.g. grain structures with binder in GrainGeo or fiber structures with binder in FiberGeo.

Start with creating a **Generation Script** (.py) using the GeoDict structure generator modules. Select the right parameter ranges to match the statistical properties for the considered materials that should be distinguished, e.g. the object diameters, the object orientations, and the solid volume percentages.

Then, with the following steps train and apply a custom neural network in GeoDict-AI:

1. Set the **Design of Experiments** defining varying parameter ranges and the number of training structures.
2. **Create** the actual **Training Data** and testing data with the defined parameter ranges.
3. Use the training data to **Train** a **Neural Network** that can identify the target material or the individual fibers in every segmented 3D-scan containing materials matching the statistical properties caught in the training data.

4. **Apply** the trained **Neural Network** on one of the generated training or testing structures to verify the results preliminary and fast.
5. **Validate** the **Performance** of the trained neural network by applying it to all generated training and testing structures.
6. Finally, use **Apply Neural Network** again to apply the trained network to segmented 3D-scans and identify the desired target material.

Another application of AI is enhancing gray value images. For example, to reduce time and costs in generating CT-scans, a neural network can be trained to enhance the image quality. For example, a scan can be taken from only a few angles or with reduced exposure time. Given low-quality / high-quality image pairs with the same resolution a neural network is trained with **Train a Neural Network**, and then applied on scans with **Enhance Grayscale Image**.

Setting up GeoDict-AI for GPUs and CPUs

In the following, the requirements for GeoDict-AI are given.

Installation

GeoDict-AI uses the TensorFlow Framework by Google, one of the most used and well-known machine learning libraries. The required version of TensorFlow is installed during the GeoDict installation and GeoDict-AI works out of the box. Install GeoDict on the used machine, to install TensorFlow correctly.

GeoDict-AI may run on the CPU (the main processor) or on the GPU (the graphics card). If a suitable GPU is detected during installation of GeoDict, GeoDict-AI will run on the GPU; otherwise it will run on the CPU.

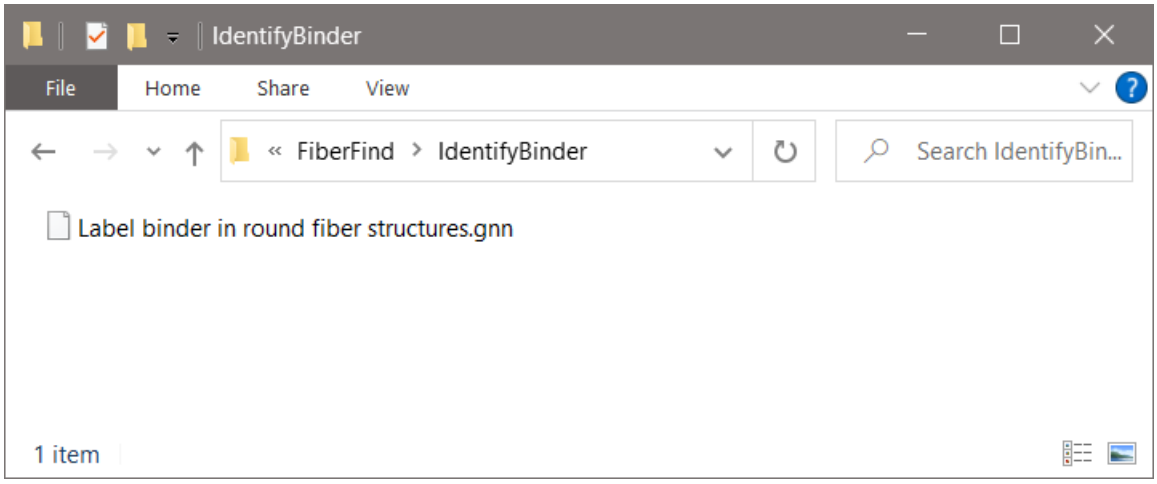
For **Linux**, the **GNU C Library** (glibc) must be at least version **2.17**. We recommend **Ubuntu 20.04 LTS**, but for a current **Gnu C Library** other Linux distributions should also work.

To ensure, that GeoDict-AI works well, a short example is provided in the FiberFind handbook to test the installation for FiberFind-AI. This test can also be done with GeoDict-AI **Apply Neural Network**, instead of FiberFind **Identify Binder (AI)**. The test needs about 5 minutes to run and shows how FiberFind-AI works and how the results of FiberFind-AI can be validated. The FiberFind handbook also explains several reasons for a failure of this test and what to do in these cases.

How to use **Apply Neural Network** is described starting on page 44. The neural network used in the test example for FiberFind can be found in the GeoDict installation folder. In Windows it is usually installed in the following path:

C:\Program Files\Math2Market GmbH\GeoDict 2023.

There, find the subfolder **IdentifyBinder** inside the folder **FiberFind** and choose the neural network **Label binder in round fiber structures.gnn**.



All other steps can be done as described in the FiberFind handbook.

USING THE GPU VERSION

The **GPU version** is usually **much faster** (roughly factor 10) than the CPU version but it requires:

- A NVIDIA graphics card (GPU) installed, with compute capability of at least 3.5.

Please make sure drivers are installed and up to date. For the version of CUDA that GeoDict 2023 is using, a version 450.80.02 or higher of the NVidia driver is needed.

See more information on graphics cards in https://developer.nvidia.com/cuda-gpus. This webpage contains a helpful section with Frequently Asked Questions.

THEORY – AI-APPROACH IN GEODICT

The **fundamental idea of AI** is that a neural network is trained to perform a special task, such as the **identification of separate fibers,** the **differentiation between material phases**, or to **enhance a grayscale image**. To train the neural network, enough examples need to be available to teach it. Clearly, for 3D-scans it is a very hard and time-consuming task to manually label materials or objects for the number of examples required. This is where **GeoDict's unique structure-generation capabilities** come in. The idea is that a neural network trained on these **synthetic** samples can then be used to analyze 3D scans of **real** materials. E.g.:

- FiberGeo can quickly generate a large number of 3D structures of fibrous media. Variation of fiber diameters, shapes, lengths, curvatures and density as well as the amount of binder, if any, can be specified to match the ranges seen in the real materials to be analyzed. Learn more about creating fibrous structures in the FiberGeo handbook of this User Guide.
- GrainGeo provides similar possibilities to generate different grain structures. Learn more about creating granular structures in the GrainGeo handbook of this User Guide.

As long as these structures are close enough to the real 3D-scans, they can be used to train neural networks using GeoDict-AI. For Enhance Grayscale Image, GeoDict can currently not generate synthetic samples. Here, the user must provide pairs of registered low-quality / high-quality (CT) scans. The neural network then learns to transform low-quality images to high-quality images. Note however that even a small number of image pairs (e.g. 2-3) can be sufficient, depending on the application.

When applied to a 3D image, the neural network will transform it to another image by assigning a scalar output value between 0 and 1 to each voxel. How this image is interpreted depends on the task at hand:

- When **identifying fibers**, the neural network will mark a centerline which corresponds to a curve along the central axis of each fiber. FiberFind-AI will then automatically reconstruct the individual fiber objects from this output.
- When **identifying material phases** (e.g. binder in a fibrous structure) the binder material is marked directly by the neural network output.
- For **enhancing gray scale images**, the neural network output corresponds directly to the improved grayscale image.

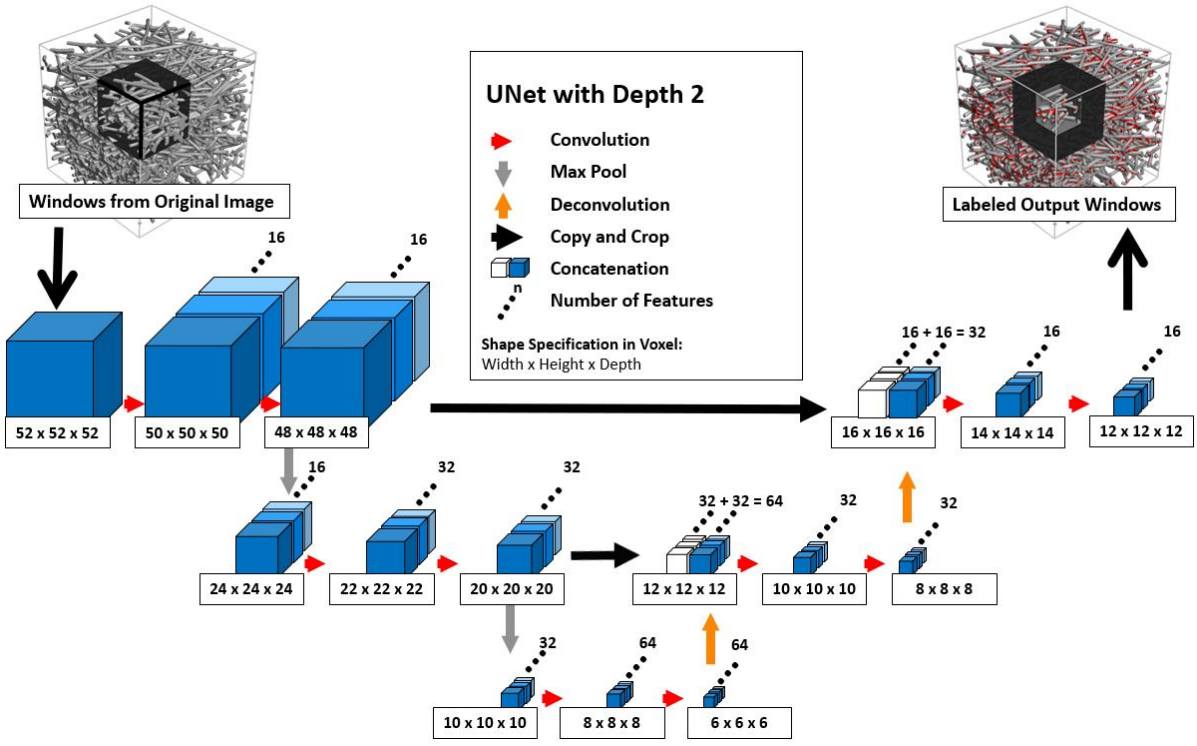
NEURAL NETWORK TRAINING

To train a neural network, GeoDict-AI uses the **UNet** method [1], which implements an image-to-image transform. When illustrated as below, the **UNet** has a U-shape (hence the name) with a convolutional constricting and a deconvolutional expanding branch on the left and right respectively. The constricting branch analyzes and simplifies the input structure to features. The expanding branch uses these features to decide, for example, which of the input voxels to label as binder.

The UNet always works on a fixed input window size and produces results for a smaller output window which is centered within the input window. In order to analyze a given 3D structure, which is usually larger than the input window, the window is automatically shifted over the whole structure and the network is applied at each window location in order to obtain results for the whole domain.

In the diagram, the given size for this input window is 52 voxels in X-, Y- and Z-direction. The window contents are encoded in the left branch of the UNet to an abstract feature map. This feature map is then decoded within the right branch of the UNet to obtain the output image.

The diagram below shows a UNet with **depth**=2. The depth is defined by the number of max pool operations / deconvolutions leading from **layer** to layer. Thus, a depth of 2 results in three layers.

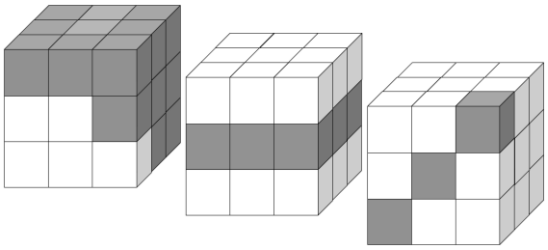


In each layer of the UNet two **convolutions** are applied on this cutout, encoding it to discriminative **features**. The first convolution in the first layer of the UNet encodes the input image into a given number of features or feature maps describing the pattern of the cutout. In the diagram above there are 16 features in the first layer.

A convolution analyzes a cutout using **kernels**. A kernel is a feature detector, i.e. a 3x3x3 voxels box with a defined pattern, for example to recognize edges. In the

figure below, three exemplary kernels are shown. The features are learned by the network, so that each network produces its own set of features.

Examples for kernels

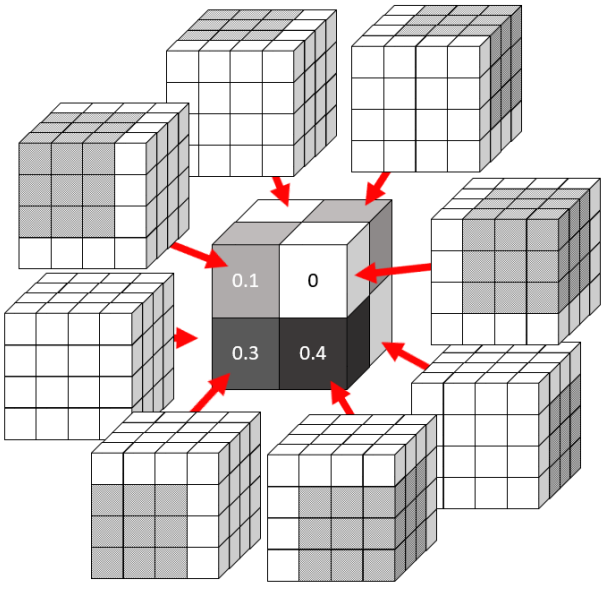


For each kernel the **convolution** then decodes the considered structure to one feature map. Therefore, a kernel scans the cutout voxel for voxel and for each 3x3x3 voxel group, it measures the similarity between the kernel and the underlying image cutout using the dot-product between the kernel values and the image values. The resulting so-called feature map indicates where the feature is found in the input image. For the given network topology, we have 16 features in the first layer, so this process is repeated for each different feature/kernel, resulting in 16 different feature maps.

As the context information from surrounding voxels has to be taken into account for the convolution, the kernel has to stay within the bounds of the input window. Thus, after each convolution the resulting cutout decreases, losing the border voxels. In the UNet diagram above, this results in a cutout size of 48x48x48 voxels after the two convolutions in the first layer.

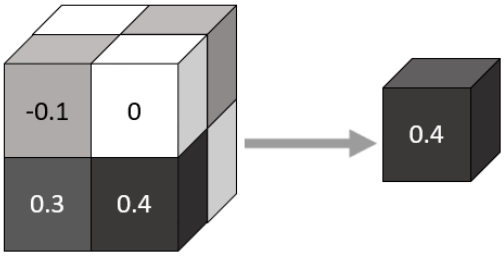
In the figure below, one kernel is applied to a 4x4x4 cutout, resulting in an 2x2x2 feature map where large values correspond to a strong detection of the feature. Regions where the feature is not present will result in low or even negative values in the feature map.

Convolution for 1 Kernel



The next layer of the UNet is then reached with a **Max Pool** operation. Max pooling halves each spatial dimension by replacing each 2x2x2 block of voxel values by a single voxel containing the maximum value within the block. In our example, this means that the second layer starts with a cutout of 24x24x24 voxels.

Max-Pool Operation



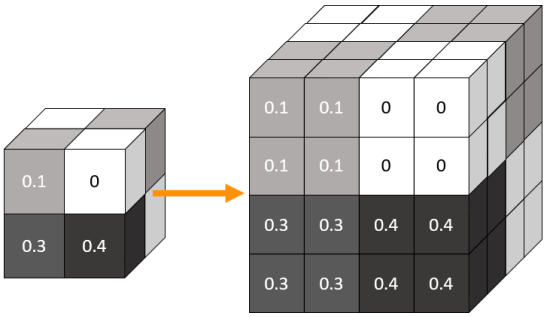
This provides the neural network with a limited form of translational invariance, meaning that it becomes tolerant to slight variations in the location of detected features at the expense of spatial accuracy. We will see later how the UNet is able to compensate for that.

In each new layer of the UNet the first convolution doubles the number of **features**. Thus, the input image is encoded into feature representations at multiple different levels. This is followed by a second convolution with the same number of output features.

At the end of the last layer, we start moving up the right (expanding) branch of the UNet, which progressively increase the resolution of the image while at the same time reducing the number of features, essentially mirroring the transformation seen in the left (constricting) branch.

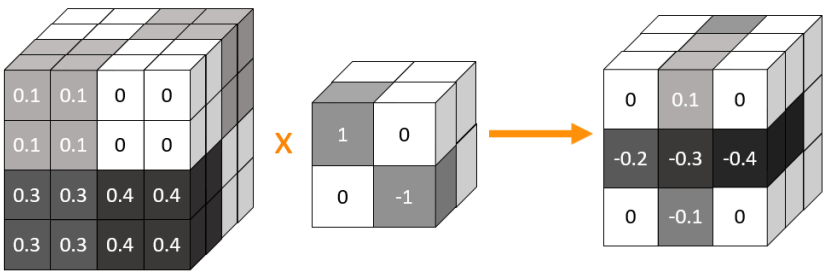
As we move up the right branch, the max-pool operation is replaced by a so-called deconvolution (transposed convolution) which doubles the spatial resolution and halves the number of feature maps. In detail, for a deconvolution first an **up-sampling** is applied on the features. Therefore, each voxel is transformed into 8 voxels with the same value, doubling the dimensions in all three directions.

Up-sampling



Then, a regular **convolution** concretizes the features, by "painting" the voxels in a feature. Here, smaller kernels are used, i.e., 2x2x2 voxels boxes with defined patterns. The feature values then are multiplied with the kernel values as shown in the figure below. Here we show an example with a kernel containing only the values -1 and 1. The kernel is applied for each 2x2x2 voxel group in the input feature map and the dot product is the value for the corresponding position in the new feature.

Convolution of up-sampled feature map



To **keep the volume dimensions**, in our case, however, zeros are appended to the input feature on three sides before multiplying it with the kernel (padding).

Afterwards, the resulting features are **concatenated** with high-resolution cutouts from the last features in the left branch in the same layer, doubling the number of feature maps. This way small features with high resolution are combined with low resolution abstract features for context information. This makes it possible to paint the voxels correctly, since it helps the neurons to build a more precise output based on this information.

Since the cutout in the right branch is still convolved twice in each layer, the final output window is smaller than the input window. In our example with a UNet of depth 2 and a window size of 52x52x52 voxels, the resulting cutout has a size of 12x12x12 voxels. This last cutout, in fact is an image again instead of a feature map.

At this point we have described the topology of the neural network, but the specific values within the convolutional kernels are not yet specified. These so-called **weights** are determined during the training process using an **optimizer** such as Adam [2]. In our supervised learning setup, we specify pairs of input and desired output images. The optimizer applies the neural network to a given input image, computes the differences between the output and the desired output (the so-called **loss**) and adjusts the weights of the neural network to reduce that error. This process is repeated iteratively until the loss converges to an acceptable value.

PREPARATION OF TRAINING MATERIAL

There are two possibilities to set-up training data. GeoDict can generate training data using its powerful structure generators. Therefore, a generation script must be created as explained starting on page 10. Then this script is used in **Design of Experiments** (page 19) and **Create Training Data** (page 25) to generate the desired amount of training data.

If other training data is available from segmented scans or a grayscale image should be enhanced, the training data must be set-up in a defined folder path, so that GeoDict recognizes it. How to do that is explained starting on page 16.

IDENTIFY TARGET MATERIAL: GENERATION SCRIPT REQUIREMENTS

GeoDict-AI is the starting point to analyze physical properties on the segmented 3D-scans considering the different materials. GeoDict-AI can be used for nearly every structure that can be generated with the GeoDict structure generator modules, e.g. grain structures with binder in GrainGeo or fiber structures in FiberGeo. To train a neural network, create structure models in GeoDict that have the same statistical properties as the scans to analyze.

Therefore, write a **Generation Script** and further use it as a mandatory input in AI-section as described on page 19. The **Generation Script** should create a series of structures. That can be also easily done with GeoDict, as far as every action in it is a command and can be saved in a GeoPy macro (*.py). A macro can be recorded by selecting **Macro → Start Macro Recording** from the menu bar, then executing the needed GeoDict commands, e.g. generating a fiber structure and adding binder, and afterwards selecting **Macro → End Macro Recording**. How to record and edit macros in GeoDict is described detailly in the Automation by Scripting handbook of this User Guide. Additionally, an example generation script can be found in the installation folder as described starting on page 26.

For the **Generation Script**, choose a structure generator in GeoDict, e.g. FiberGeo or GrainGeo, and generate a structure matching the scan parameters, while recording it in a macro. Then, add the parameters to vary in the **Variables** section.

The generation script must, at the very least, contain a variable called **gdSeed**. When generating structures for training, the value of **gdSeed** is automatically varied from one structure to the next. The script must also pass this variable along to the structure generators being used in the script, as described below. Otherwise, all generated structures will be the same.

The variables section can also have more parameters. In the example, a neural network is trained to identify binder in a fiber structure with varying solid volume percentage of fibers and binder. Therefore, variables are added for solid volume percentage and binder solid volume percentage in the **Variables** section.

```
Variables = {
  'NumberOfVariables' : 1,
  'Variable1' : {
    'Name'            : 'gdSeed',
    'Label'           : 'Random Seed',
    'Type'            : 'int',
    },
  }
```

Here, only variable 2 uses all possible keys. For example, the built-in default value can be omitted, as it is done for variables 1 and 3. This value is not needed for the training. It is only shown in the **Macro Execution Control**, when executing a parametric macro as described in the Automation by Scripting handbook.

```
Variables = {
  'NumberOfVariables' : 3,
  'Variable1' : {
    'Name'           : 'gdSeed',
    'Label'          : 'Random Seed',
    'Type'           : 'int',
    },
  'Variable2' : {
    'Name'           : 'gd_SVP',
    'Label'          : 'Solid Volume Percentage',
    'Type'           : 'double',
    'Unit'           : '%',
    'BuiltinDefault' : 10.0,
    'Check'          : 'min0;max100'
    },
  'Variable3' : {
    'Name'           : 'gd_BinderSVP',
    'Label'          : 'Binder SVP',
    'Type'           : 'double',
    'Unit'           : '%',
    'Check'          : 'min0;max100'
    },
  }
```

The parameters defined in the **Variables** section, then need to be inserted at the right places in the corresponding dictionaries.

The variable **gdSeed** is entered as a value for **RandomSeed**, which is a parameter of most structure generators in GeoDict.

```
Create_args_1 = {
  'MaterialMode'              : 'Material',
  'MaterialIDMode'            : 'MaterialIDPerObjectType',
  'Domain' : {
  'MaximalTime'               : (24, 'h'),
  'OverlapMode'               : 'RemoveOverlap',
  'NumberOfObjects'           : 100,
  'StoppingCriterion'         : 'SolidVolumePercentage',
  'SolidVolumePercentage'     : (gd_SVP, '%'),
  'Grammage'                  : (10, 'g/m^2'),
  'SaveGadStep'               : 10,
  'RecordIntermediateResult'  : False,
  'InExisting'                : False,
  'KeepStructure'             : False,
  'WeightPercentage'          : (0, '%'),
  'Density'                   : (0, 'g/cm^3'),
  'RemoveOverlap' : {
  'MatchSVFDistribution' : {
  'PercentageType'            : 0,
  'RandomSeed'                : gdSeed,
  'IsolationDistance'         : (0, 'm'),
  'ResultFileName'            : 'FiberGeo.gdr',
  'MatrixDensity'             : (0, 'g/cm^3'),
  'OverlapMaterial' : {
  'ContactMaterial' : {
  'NumberOfGenerators'        : 2,
  'Generator1' : {
  'Generator2' : {
  'Temperature'               : (293.15, 'K'),
  }
gd.runCmd("FiberGeo:Create", Create_args_1, Header['Release'])
```

```
AddBinder_args_1 = {
  'ResultFileName'         : 'Binder.gdr',
  'MaterialMode'           : 'Material',
  'BinderMaterial' : {
  'ContactAngle'           : (0, 'Deg'),
  'StoppingCriterion'      : 'SolidVolumePercentage',
  'SolidVolumePercentage'  : (gd_BinderSVP, '%'),
  'Grammage'               : (10, 'g/m^2'),
  'WeightPercentage'       : (20, '%'),
  'MaterialDensity'        : (1, 'g/cm^3'),
  'BinderDensity'          : (2.70009, 'g/cm^3'),
  'PeriodicX'              : False,
  'PeriodicY'              : False,
  'PeriodicZ'              : False,
  'UseTolerance'           : True,
  'UseMaxIterations'       : False,
  'UseMaxTime'             : False,
  'Tolerance'              : 0.05,
  'MaxNumberOfIterations'  : 10,
  'BinderAnisotropy'       : 1,
  'MaxTime'                : (3600, 's'),
  'DistributionMode'       : 'Homogeneous',
  'LayerDensities'         : [0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75,
  'Temperature'            : (293.15, 'K'),
  'SVPCorrection'          : False,
  }
gd.runCmd("FiberGeo:AddBinder", AddBinder_args_1, Header['Release'])
```

The variable **gdSeed** then is used to set up the random number generation for the structure generator. Additionally, if **Random** was selected for **Variation** in the **Design of Experiments** dialog, **gdSeed** is used to set up the random number generation for the other parameters defined in the Variables section.

The requirements for the output of the **Generation Script** vary depending on the postprocessing mode selected. For fiber identification, at the end of the script there should be a fiber structure in memory, including the analytic object representation of the fibers (GAD data). This requirement is fulfilled for the previous example which generates a fiber structure. Adding binder material also keeps the GAD information about the fibers intact. Note that not all operations will preserve GAD data. To learn more about GAD objects, refer to the GadGeo handbook of this User Guide.

When distinguishing **Material Phases** instead as described on page 25, the generation script must explicitly write a pair of *.gdt files (GeoDict structure files) as follows:

- **input.gdt**: In this structure all solid material phases must be assigned to material ID 01.
- **output.gdt:** In this structure the target material must have material ID 02 and all other solid materials must be assigned to ID 01.

Therefore, reassign all solid material IDs, that are NOT the target material ID, to 01. Thus, there are only three material IDs in the output.gdt file,

- **00** for the pore space,
- **01** for the solid materials without the target material,
- **02** for the target material.

For example, consider a **Generation Script** which produces a fiber structure with binder where material ID 00 is pore space, material IDs 01 and 02 are fibers and Material ID 03 is the added binder material phase. To complete the script for GeoDict-AI, the correct material IDs must be assigned and the two output structures saved as follows:

1. First material ID 02 must be reassigned to material ID 01 with ProcessGeo. This option can be found when selecting **Model → ProcessGeo** from the menu bar and selecting **Reassign** from the pull-down menu in the **ProcessGeo** section. When recording the command in a macro, it looks as follows:

```
Reassign_args_1 = {
   'AssignMaterial' : False,
   'SwapIDs'        : False,
   'OldMaterialID' : 2,
   'NewMaterialID' : 1,
   }
gd.runCmd("ProcessGeo:Reassign", Reassign_args_1, Header['Release'])
```

2. Afterwards, using the same command, reassign the target material, in our example the binder, to material ID 02.
3. Then, save the resulting structure as **output.gdt** (**File → Save Structure as...**).

```
SaveFile_args_1 = {
   'FileName'    : 'output.gdt',
   'FileVersion' : 3,
   }
gd.runCmd("GeoDict:SaveFile", SaveFile_args_1, Header['Release'])
```

   The key 'FileVersion' defines the file format and the value 3 means *.gdt, which is the default structure file format since GeoDict 2023. Thus, the command will look as shown above when it is recorded in a macro.

4. For the **input.gdt** file reassign the target material also to material ID 01 to obtain a structure, with only pore space and solid material. Then save the structure as input.gdt (**File → Save Structure as...**).

```
SaveFile_args_1 = {
   'FileName'    : 'input.gdt',
   'FileVersion' : 3,
   }
gd.runCmd("GeoDict:SaveFile", SaveFile_args_1, Header['Release'])
```

All these commands do not need to be typed in the text editor but can be recorded directly from GeoDict as described in the Automation by Scripting handbook of this User Guide.

Especially for structures containing binder it can be necessary to generate the structures bigger than intended and then crop them with **Model → ProcessGeo → Crop** to prevent boundary effects before saving the structures to output.gdt and input.gdt.

Checklist for Structure Parameters

The parameters for structure generation must be chosen to match the properties of the target scan. Ensure to cover the complete required parameter space, by capturing all variations observed in the target materials.

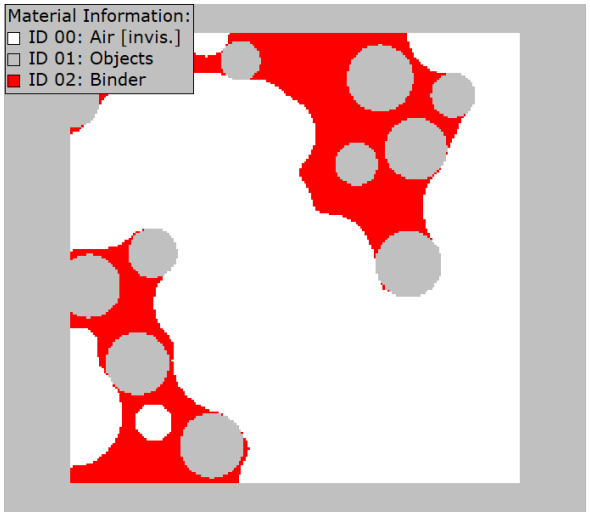
In the following a parameter checklist is given:

- Voxel length (resolution)
- Solid volume fractions for the different materials
- Object diameter ranges
- Object shapes
- Object orientation distribution
- Object overlap (set overlap not larger as is necessary to represent artefacts in the scan)
- Fiber curvature
- CT artefacts (e. g. hour glassing and fiber crossings)
- Contact angle of binder
- Binder anisotropy
- …

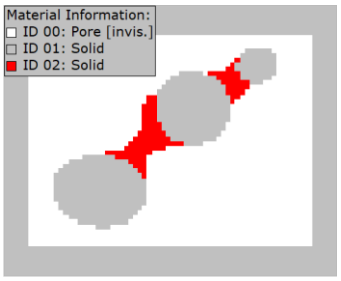
The numbers of voxels in the X-, Y- and Z-direction define the **Domain Size** and should already consider the **Window Sizes** (explained on page 34) used for training. For example, for the default window size of 52^3 voxels, a domain size between 256^3 and 512^3 voxels is recommended. Larger structures require higher hardware resources.

Make sure to **check on boundary effects** when generating structures. In some cases, it can be necessary to crop the structure with **ProcessGeo** → **Crop** to avoid them.

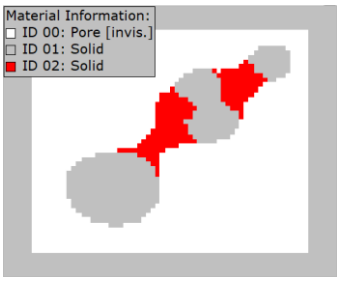
Usually, if the different material phases or the individual fibers can be **recognized by eye**, the neural network can also learn to distinguish them. Otherwise, if for example the binder solid volume percentage is very high compared to the object diameters, the objects can "disappear" inside the binder. Then, it is nearly impossible for the network to identify the small objects inside the binder.



Avoid large parameter ranges, since the training of the neural network would become very difficult. Consider for example a fiber structure with binder. If there are fibers with very different diameters, probably many of the thicker fibers will be recognized as two thinner fibers with binder in between and the other way around.

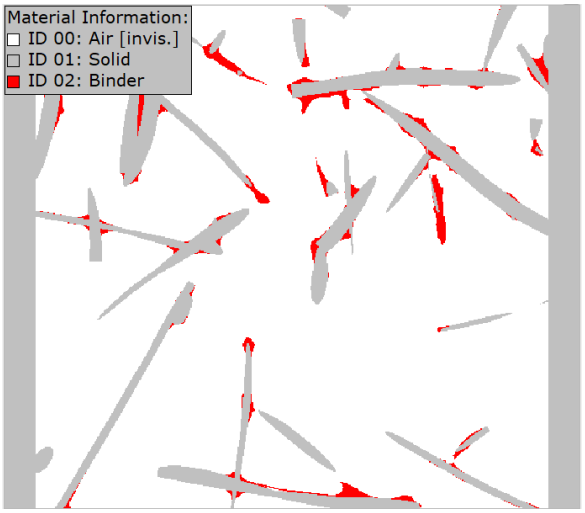


Ground truth

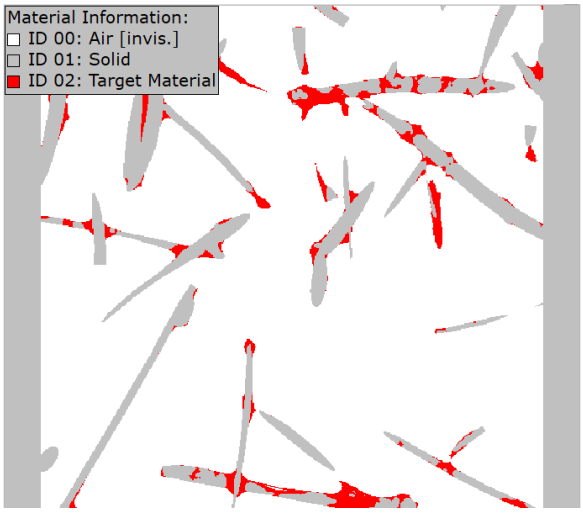


Output of neural network

Make sure to **remove the overlap**, only leaving the amount of overlap that is necessary to represent artefacts in the scan.



Ground truth



Output of neural network

Enhance Grayscale Image Training Data Requirements

While ImportGeo-Vol has many effective image filters, as explained in the ImportGeo-Vol handbook of this User Guide, there are cases, where a neural network performs better. Use GeoDict-AI to train a neural network to enhance grayscale images, for example for CT-scans taken from only a few angles or with reduced exposure time. The trained neural network can either be used with **Enhance Grayscale Image** described on page 54 or in the ImportGeo-Vol **Image Processing** dialog.

To train the neural network, you need low-quality / high-quality image pairs.

These must fulfill the following requirements:

- they must be registered, i.e. capture the same region,
- currently, they must have the same resolution and therefore, the same number of voxels in X-, Y- and Z-direction,
- the low-quality scan should be saved as input.grw and the high-quality scan should be named output.grw.

The neural network will then learn to transform low-quality scans to high-quality scans such that ideally you will be able to obtain comparable high-quality results by applying the trained network to future low-quality scans.

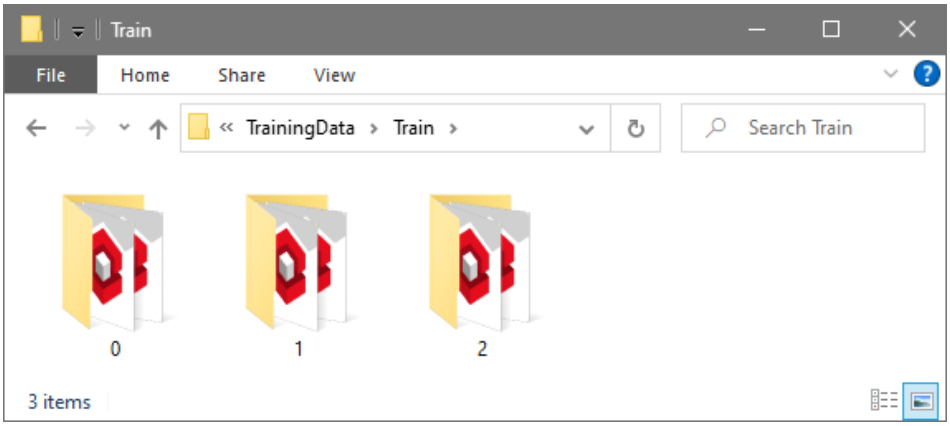
Therefore, the files must be organized in the folder structure described below.

Set-up Training Data manually

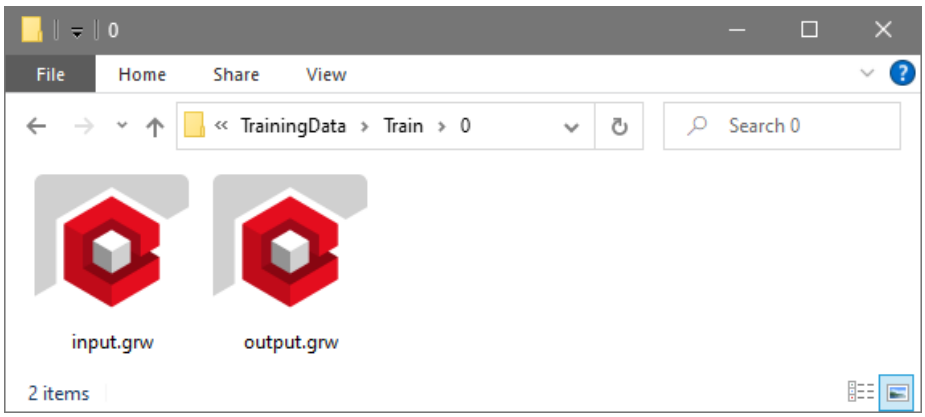
For the training the available input and output files have to be saved in a defined folder structure. While for structure files (*.gdt) generated with the GeoDict structure generator using **Create Training Data** the required folder structure is produced automatically, it has to be created manually, if the training data is produced outside of GeoDict.

Create an empty folder for the training data. The name of this folder can be chosen freely according to the corresponding project. In the user guide example, it is called "TrainingData".

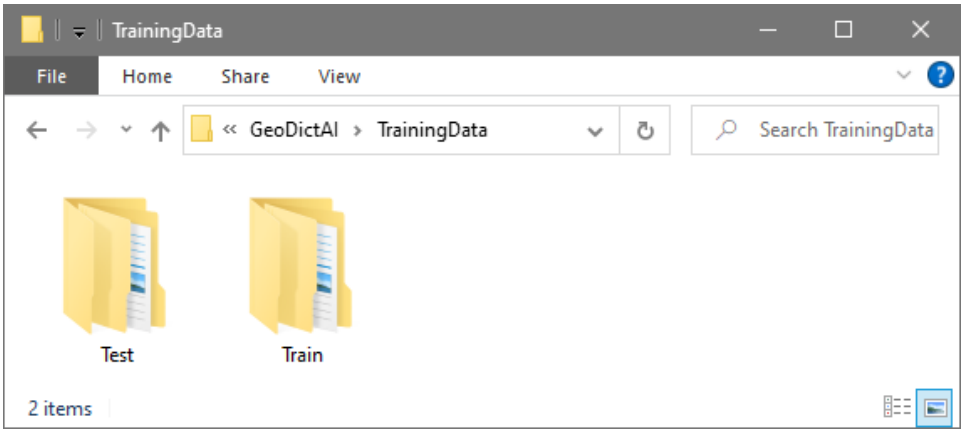
Inside this folder create a folder named **Train**. For each training pair (input.grw and output.grw or input.gdt and output.gdt) create a folder, which can again be named freely, for example with the scan's name. In the example we called the folders as it is done when creating training data automatically with numbered folders starting with "0".



The neural network will learn how to transform an input.grw (input.gdt) into an output.grw (output.gdt). Thus, place the input.grw (input.gdt) and output.grw (output.gdt) pairs in the folders.



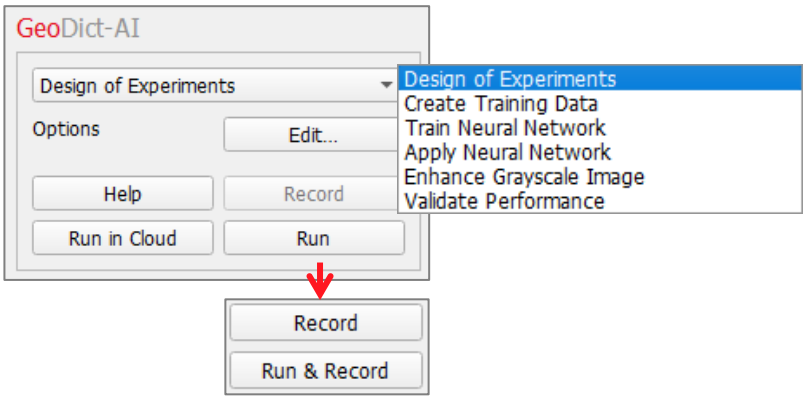
For identifying fibers and distinguishing between material phases, additionally a folder **Test** can be placed inside the training data folder containing more folders with input.gdt and output.gdt pairs, if enough pairs are available. This data can be used in **Validate Performance** described starting on page 58 to verify if the network performs well on structures not included in the training.



GeoDict-AI Section

GeoDict-AI starts when selecting **Analyze** → **GeoDict-AI** in the menu bar.

Five processes are listed and can be selected from the pull-down menu in the **GeoDict-AI** section: **Design of Experiments**, **Create Training Data**, **Train Neural Network**, **Apply Neural Network**, **Enhance Grayscale Image** and **Validate Performance**.



The options for each process can be edited after selecting it from the pull-down menu and clicking the **Options' Edit...** button.

When the options for the selected GeoDict-AI process have been entered, clicking **Run** starts the computation.

When recording a macro, e.g. to run parameter studies, the **Record** button becomes active and the **Run** button changes to **Run & Record**.

Click **Run in Cloud** to run it in the GeoDict cloud, see the High Performance Computing handbook of the GeoDict User Guide for details. If interested in cloud simulations, contact Math2Market to apply for a GeoDict cloud license.

While **Create Training Data** only generates the training and testing structures placing them in the selected folder, all other results from the GeoDict-AI computations are saved as ***.gdr** files in the project folder.

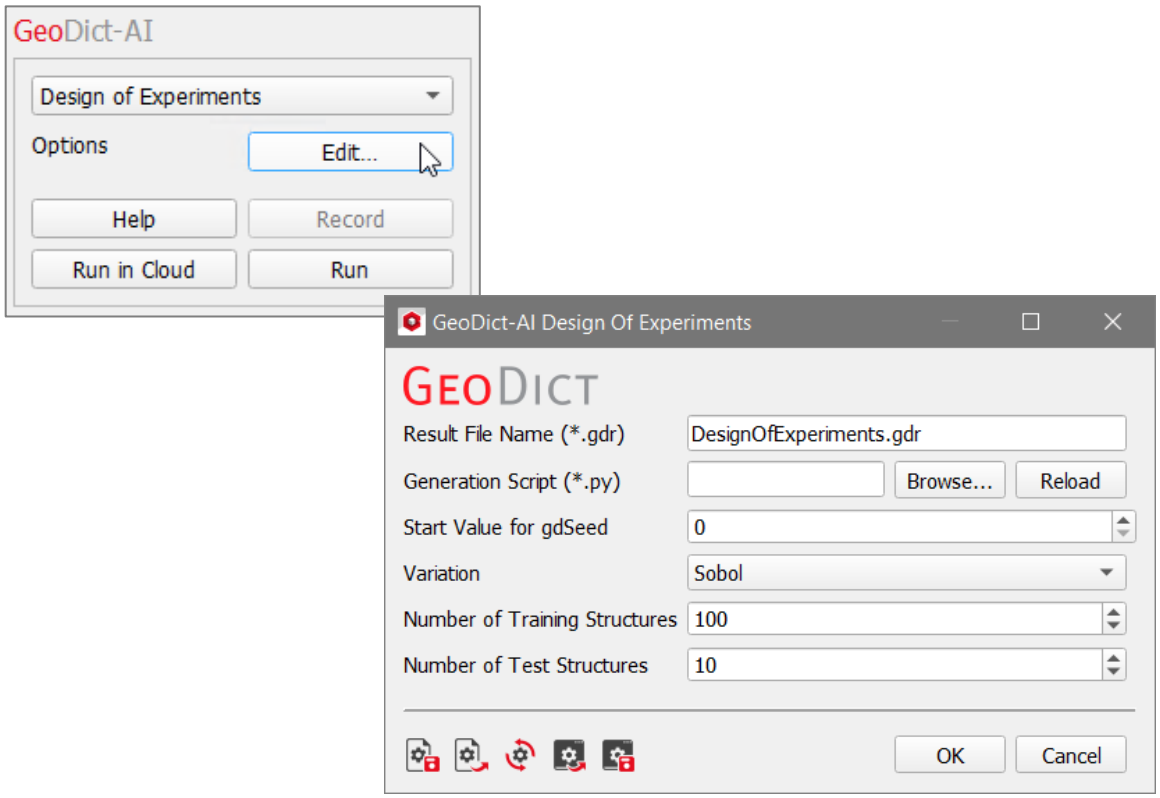
GeoDict result files (.gdr files) can be opened with the GeoDict Result Viewer at any time by selecting **File** → **Open Results (*.gdr)** from the Menu bar.

If you save the **Options** dialogs parameters into GPS (**GeoDict** Project Settings) files, you can reload them at will.

Remember to restore and reset your (or GeoDict's) default values through the icons at the bottom of the dialogs when needed and/or before every GeoDict-AI run. Rest the mouse pointer over an icon to see a tooltip showing the icon's function.

Design of Experiments

The training of a neural network usually begins with setting the experiment parameters. Therefore, select **Design of Experiments** from the pull-down menu in the GeoDict-AI section. The **GeoDict-AI Design of Experiments Options** dialog opens when clicking the **Options' Edit...** button



At the top of the **GeoDict-AI Design of Experiments** dialog, the name for the files containing the design results can be entered for **Result File Name (*.gdr)**. The default name can be kept, or a new name can be chosen fitting the current project.

Generation Script and Generation Script Parameters

For **Generation Script**, browse to the *.py file containing the information to generate the training data. How to create this GeoPy file is explained starting on page 10.

The generation script must contain the parameter **gdSeed**. It should be used, to provide a random seed to any stochastic structure generation commands (e.g. FiberGeo or GrainGeo). Enter a **Start Value for gdSeed**. Starting with the entered value, with each structure generation gdSeed is counted one up.

After selecting a generation script, the **Generation Script Parameters** panel appears below the gdSeed value. If the generation script contains variables other than the gdSeed parameter, here, the allowed value ranges for these parameters can be given in the **min** and **max** parameter boxes.

In the example below, the selected **Generation Script** contains the parameters gdSeed, gd_SVP and gd_BinderSVP. According to the entered parameters, the solid volume percentage will be varied between 30% and 40%, and the binder solid volume percentage will vary between 1.5 and 2.5.

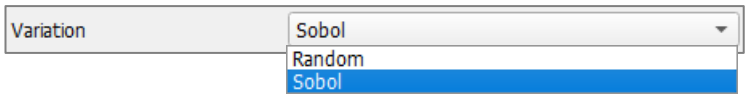
```
Variables = {
  'NumberOfVariables' : 3,
  'Variable1' : {
    'Name'           : 'gdSeed',
    'Label'          : 'Random Seed',
    'Type'           : 'int',
    },
  'Variable2' : {
    'Name'           : 'gd_SVP',
    'Label'          : 'Solid Volume Percentage',
    'Type'           : 'double',
    'Unit'           : '%',
    'BuiltinDefault' : 10.0,
    'Check'          : 'min0;max100'
    },
  'Variable3' : {
    'Name'           : 'gd_BinderSVP',
    'Label'          : 'Binder SVP',
    'Type'           : 'dou
    'Unit'           : '%',
    'Check'          : 'min
    },
  }
```

Generation Script Parameters

	min	max
Solid Volume Percentage / (%)	30	40
Binder SVP / (%)	1.5	2.5

VARIATION

Select a **Variation** method for the script parameters to determine how the parameters are chosen from the entered value ranges. The available number generation methods are **Sobol** and **Random**.



Random will draw independent random samples in the given parameter ranges. The start value for gdSeed will be used to initialize the random number generation. While all other settings are left the same, the same **gdSeed** start value always generates the same parameter sequences. Accordingly, different seeds generate different parameter sequences.

Sobol uses a so-called sobol sequence for sampling which generally results in more uniform covering of the n-dimensional parameter space. **Sobol** does not depend on gdSeed. To learn more about this algorithm refer to [4].

For a comparison between these two **Variation** methods see page 23.

NUMBER OF TRAINING AND TEST STRUCTURES

The **Training Structures** will be used later to train a neural network. For each parameter combination an input and an output file are generated. The neural network will learn how to transform the input file in the given output file. Learn more about input and output files see page 27. Select the **Number of Training Structures**. The more structures used, the more the neural network can learn in the training. The minimum number needed depends on the number of varying structure parameters but in most cases 100 training structures lead to good results.

The **Test Structures** can be used to validate a neural network, that was trained on the training structures. They are created with the same generation script but with different parameter combinations within the given boundaries. If the neural network

performs well on these test structures, it should also deliver good results on segmented scans with the corresponding statistical properties. The performance is validated, by applying the trained network to the input files and comparing the results to the corresponding output files. Learn more about input and output files see page 27. Select the **Number of Test Structures**. The more structures used, the better for the neural network validation. In most cases with 10 test structures the network can be validated well. How to **Validate Performance** see page 58.

However, for testing of the generation script it is recommended to choose a small **Number of Training** and **Test Structures**, e.g. 1-5.

If all parameters are set as desired, click **OK** to close the dialog and in the GeoDict-AI section click **Run**.

RESULTS

Running **Design of Experiments** produces a GeoDict result file (*.gdr) and a result folder with the same name. Both are saved in the chosen project folder (**File → Choose Project Folder...** in the menu bar).

The result folder contains the **samples.csv** file. This is the file later used to **Create Training Data**, as described on page 25, and allows to check, if the parameter combinations produced from the given parameter ranges are reasonable for the desired training data, before starting the time-consuming generation process.

The **samples.csv** file can also be opened in Excel or a text editor to review the content.

samples.csv

```
Type,gdSeed,gd_SVP,gd_BinderSVP
Train,5,25,2
Train,6,27.5,1.75
Train,7,22.5,2.25
Train,8,23.75,1.875
Train,9,28.75,2.375
Train,10,26.25,1.625
Train,11,21.25,2.125
Train,12,21.875,1.8125
Train,13,26.875,2.3125
Train,14,29.375,1.5625
Test,15,24.375,2.0625
Test,16,23.125,1.6875
Test,17,28.125,2.1875
Test,18,25.625,1.9375
Test,19,20.625,2.4375
Test,20,20.9375,1.96875
Test,21,25.9375,2.46875
Test,22,28.4375,1.71875
Test,23,23.4375,2.21875
Test,24,24.6875,1.59375
```

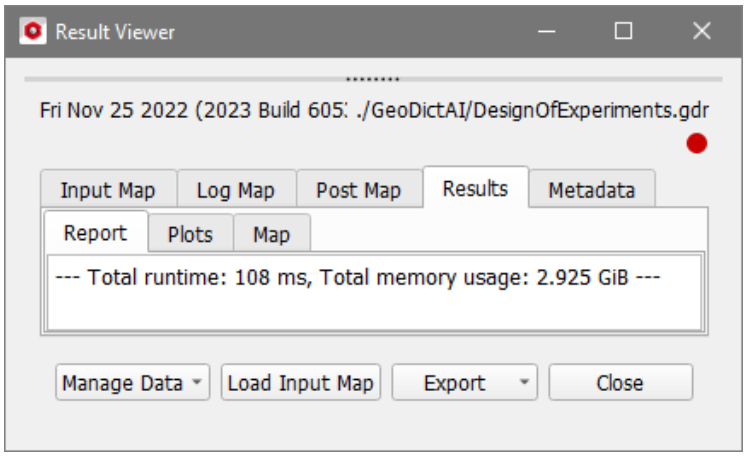
	A	B	C	D
1	Type	gdSeed	gd_SVP	gd_BinderSVP
2	Train	5	25	2
3	Train	6	27.5	1.75
4	Train	7	22.5	2.25
5	Train	8	23.75	1.875
6	Train	9	28.75	2.375
7	Train	10	26.25	1.625
8	Train	11	21.25	2.125
9	Train	12	21.875	1.8125
10	Train	13	26.875	2.3125
11	Train	14	29.375	1.5625
12	Test	15	24.375	2.0625
13	Test	16	23.125	1.6875
14	Test	17	28.125	2.1875
15	Test	18	25.625	1.9375
16	Test	19	20.625	2.4375
17	Test	20	20.9375	1.96875
18	Test	21	25.9375	2.46875
19	Test	22	28.4375	1.71875
20	Test	23	23.4375	2.21875
21	Test	24	24.6875	1.59375

The first column in the **samples.csv** always contains the **Type** for the generated value combinations. The possible types are **Train** and **Test**. The number of rows then depends on the **Number of Training Structures** (**Train**) and the **Number of Test Structures** (**Test**) entered in the **Design of Experiments Options** dialog.

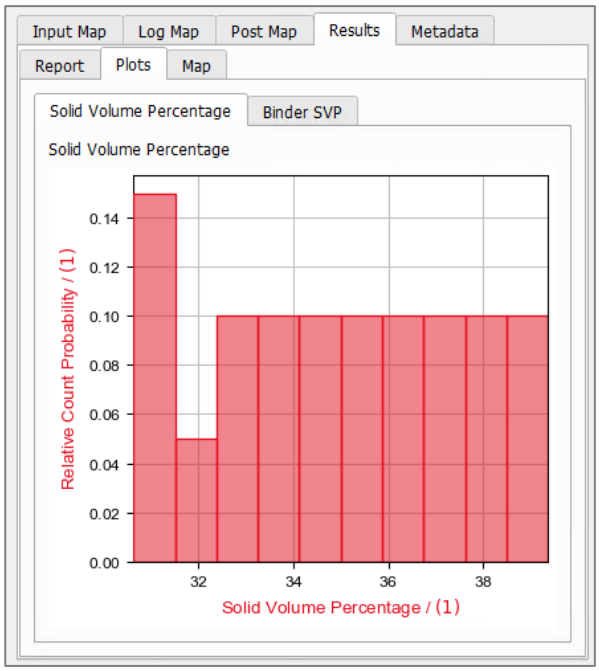
In the second column the values for **gdSeed** can be found starting with the **Start Value for gdSeed**.

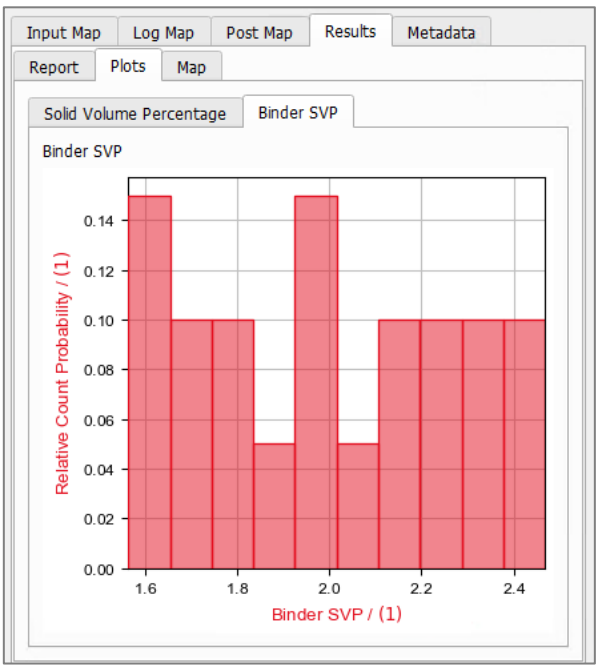
If there are also other parameters in the **Generation Script**, a column with random values in the given ranges is produced for each of them.

The GeoDict **Result Viewer** opens for the result file and only the performance of the experiment design is shown in the **Results – Report** subtab.



If the **Generation Script** contains other parameters besides **gdSeed**, the **Results – Plot** subtab provides plots for all these parameters. The plots show the **Relative Count Probability** for each value in the given ranges.

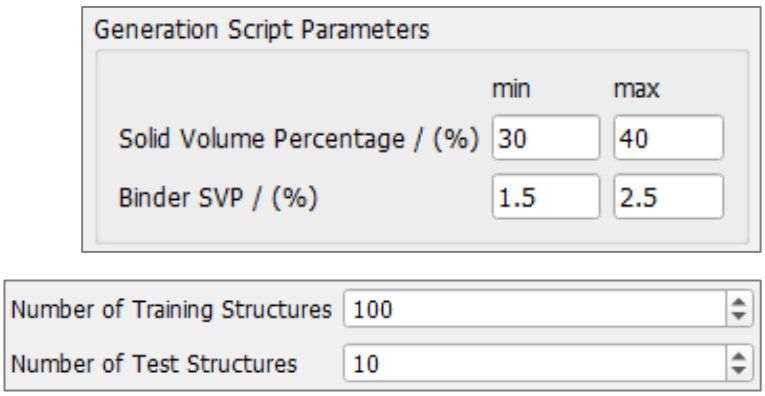




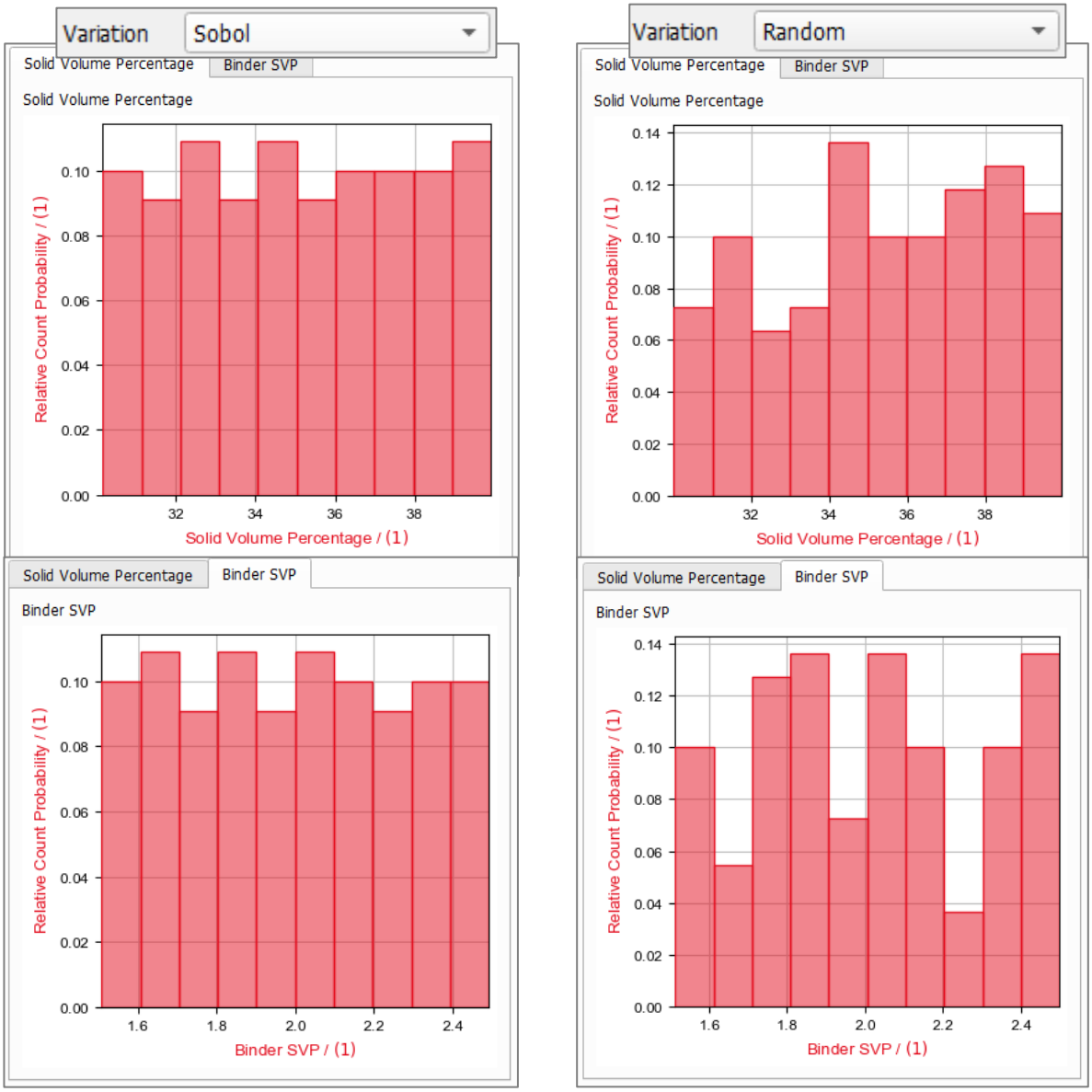
In the following example, observe the differences between the two **Variation** options **Sobol** and **Random**.

The example script had three parameters. The **Start Value for gdSeed** was set to 5 and the parameter ranges were set to 30-40 for **Solid Volume Percentage** and 1.5 – 2.5 for **Binder SVP**.

The **Number of Training Structures** was set to 100 and the **Number of Test Structures** were set to 10.

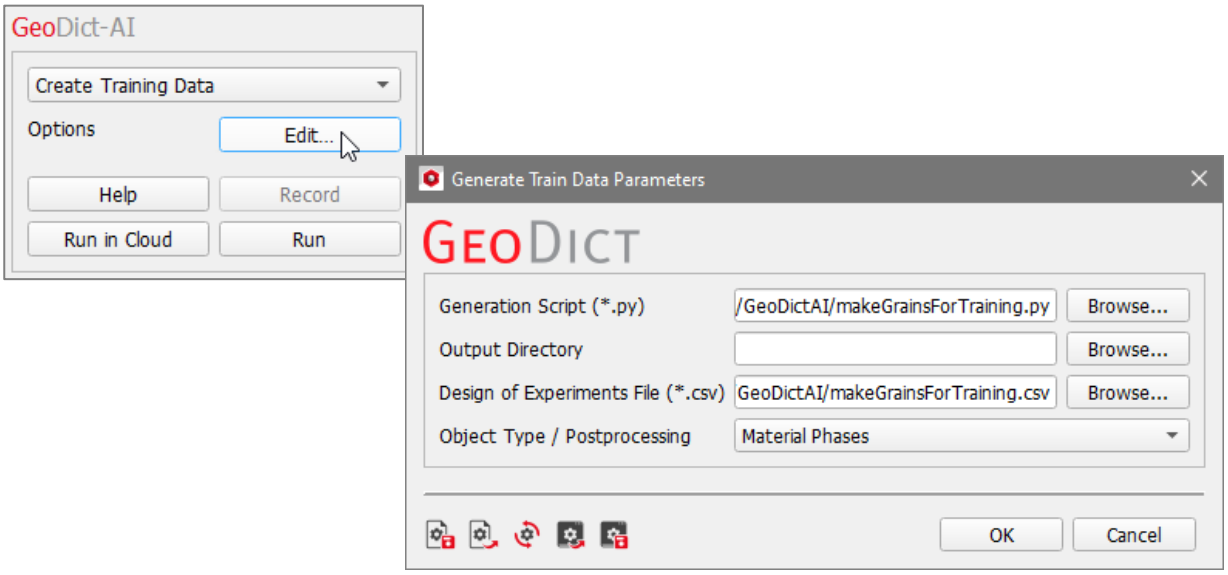


This results in the following plots for the two different **Variation** options. Observe how the parameter values are selected uniformly in the given ranges, if **Sobol** is selected, while they are chosen randomly if **Random** is selected.



Create Training Data

To create training data in the GeoDict-AI section select **Create Training Data** from the pull-down menu. The **Generate Train Data Parameters** dialog opens when clicking the **Options' Edit...** button in the GeoDict-AI section.



Generation Script, Output Directory and Design of Experiments File

For **Generation Script**, browse to the *.py file containing the information to generate the training data. How to create this GeoPy file is explained starting on page 10. Use the same file as for the corresponding **Design of Experiments** described on page 19.

For the **Output Directory**, browse to an empty folder to store the resulting training data or create a new folder. The folder must be empty, because it is used to train the neural network, as described on page 33.

Browse for the **Design of Experiments File (*.csv)** created to set up the experiment parameter ranges and number of training and test structures (see page 22).

Object Type / Postprocessing

Three different **Object Types** can be selected to define the **Postprocessing**.

Select **Material Phases** if only two material phases should be distinguished, e.g. grains and binder or circular fibers and rosetta fibers. Therefore, the **Generation Script** must produce the files **input.gdt** and **output.gdt** as described starting on page 12.

Choose **Fibers (centerlines)** if the individual fibers should be identified. With this option, **Create Training Data** will perform a postprocessing, identifying the fiber's centerlines and producing the files input.gdt and output.gdt, automatically.

To identify individual grains, use the GrainFind module. There, grains are identified using the watershed algorithm as described in the GrainFind handbook of this User Guide.

If all parameters are set as desired, click **OK** to close the dialog and in the GeoDict-AI section click **Run**.

Default Example

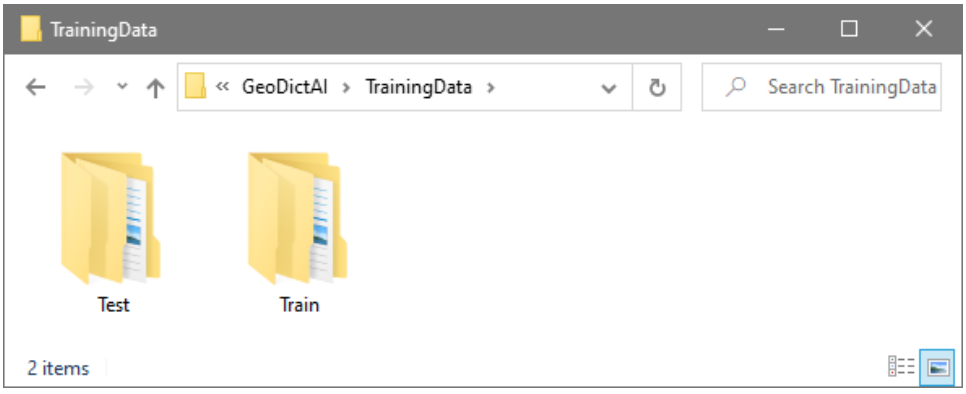
By default, for **Generation Script** and **Design of Experiments File**, two example files are loaded. Find the files **makeGrainsForTraining.py** and **makeGrainsForTraning.csv** files in the GeoDict installation folder. Browse for the GeoDict **installation folder** and find the files inside the **GeoDictAI** folder.

The generation script example generates a grain structure with overlap, adds binder and saves **input.gdt** and **output.gdt.** Open **makeGrainsForTraining.py** in a text editor to examine the structure of the example **Generation Script**.

The training settings, as the variable values and number of training structures are controlled by the **makeGrainsForTraining.csv** file. It can be opened in Excel or another text editor.

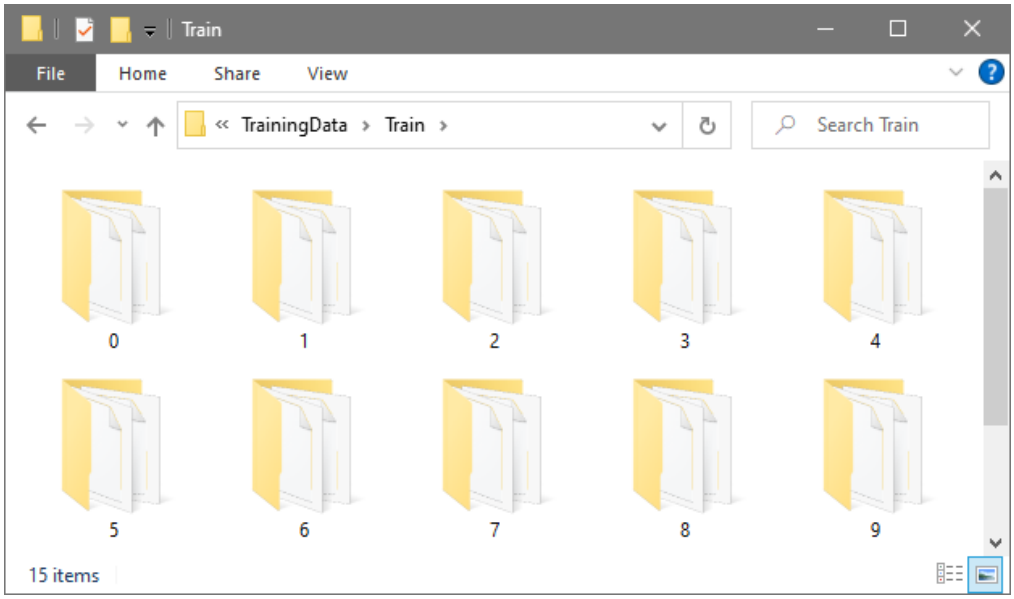
Results

First the **Training Structures** are generated, followed by the **Test Structures**. They are placed in the subfolders **Train** and **Test** inside the selected **Output Directory**.



For the training only the folder **Train** will be used. The folder **Test** can be used to validate the performance of the network.

Each of the subfolders (e.g., 'Train/0/') contains the results produced by the **Generation Script** and a pair of GDT files called **input.gdt** and **output.gdt**.

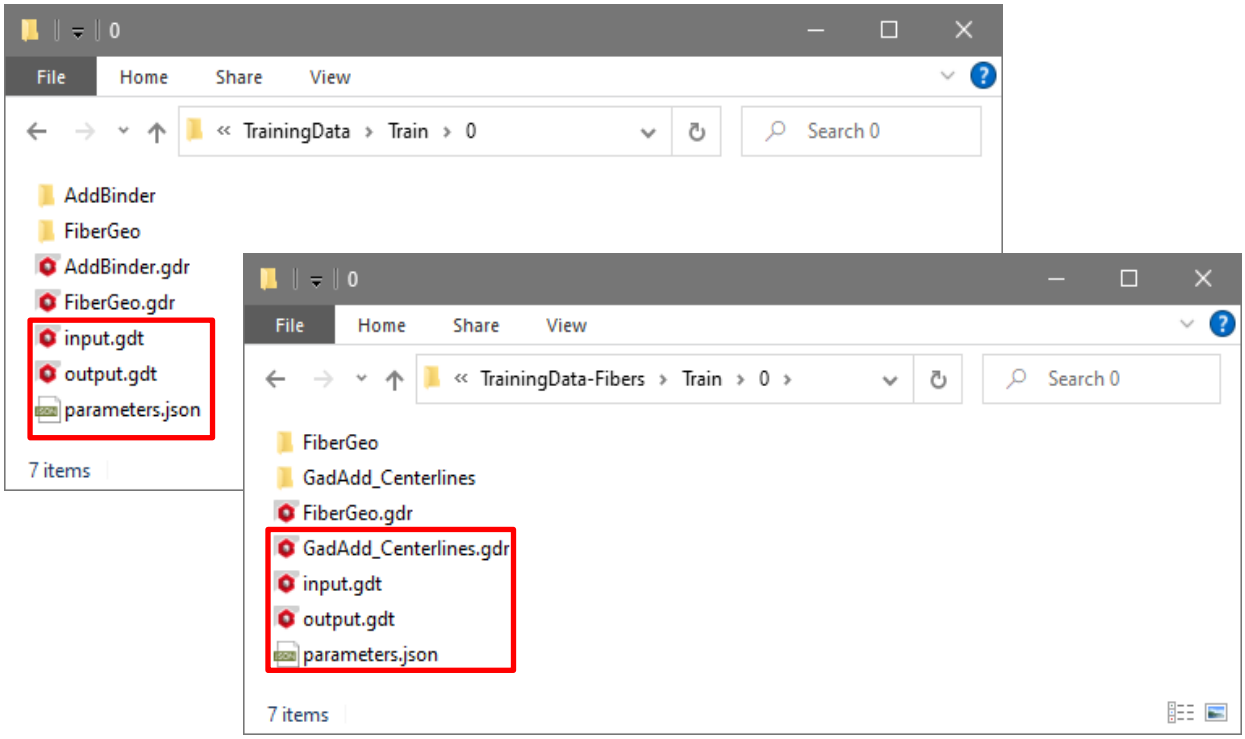


If **Material Phases** is selected for **Object Type / Postprocessing**, the input and output files must be generated by the **Generation Script**. In the **input.gdt** then, the **Target Material** cannot be distinguished, while in the **output.gdt**, the **Target Material** has its own material ID.

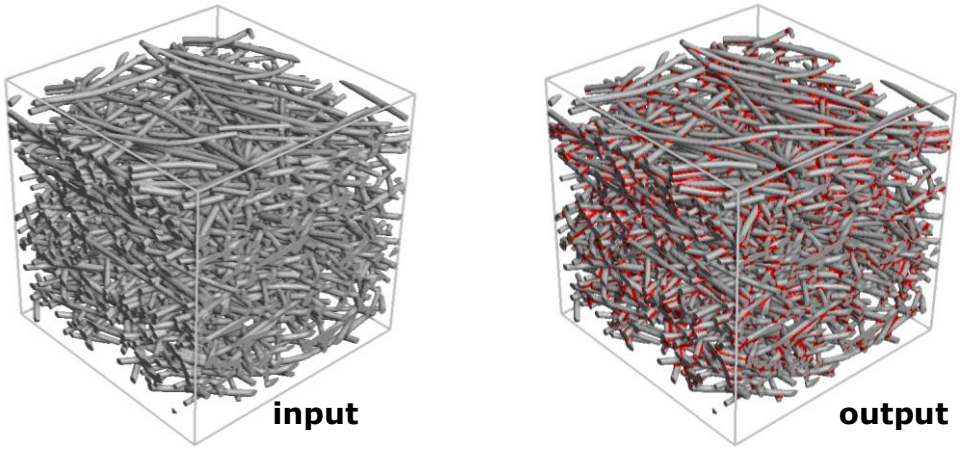
The two files are generated automatically if **Fibers (Centerlines)** is selected instead. Then, the **input.gdt** contains only the fibers, which are then assigned to material ID 01, while in the **output.gdt** the centerlines in the fibers are assigned to material ID 02.

Later in training, in both cases, the neural network learns how to produce an output from an input file by assigning those voxels with the corresponding material ID.

The macro parameters for the training structure are in the **parameters.json** file.



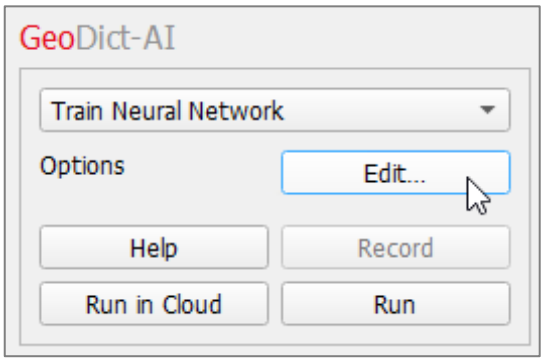
To verify that the input and output files are as intended, load them to GeoDict by selecting **File → Open Structure (*.gdt, *.gad) ...** from the menu bar.



Alternatively, if input and output files are not generated with a Generation script, create the folder structure manually and place the input.gdt and output.gdt in subfolders folders inside the Train and Test folders, as described starting on page 16.

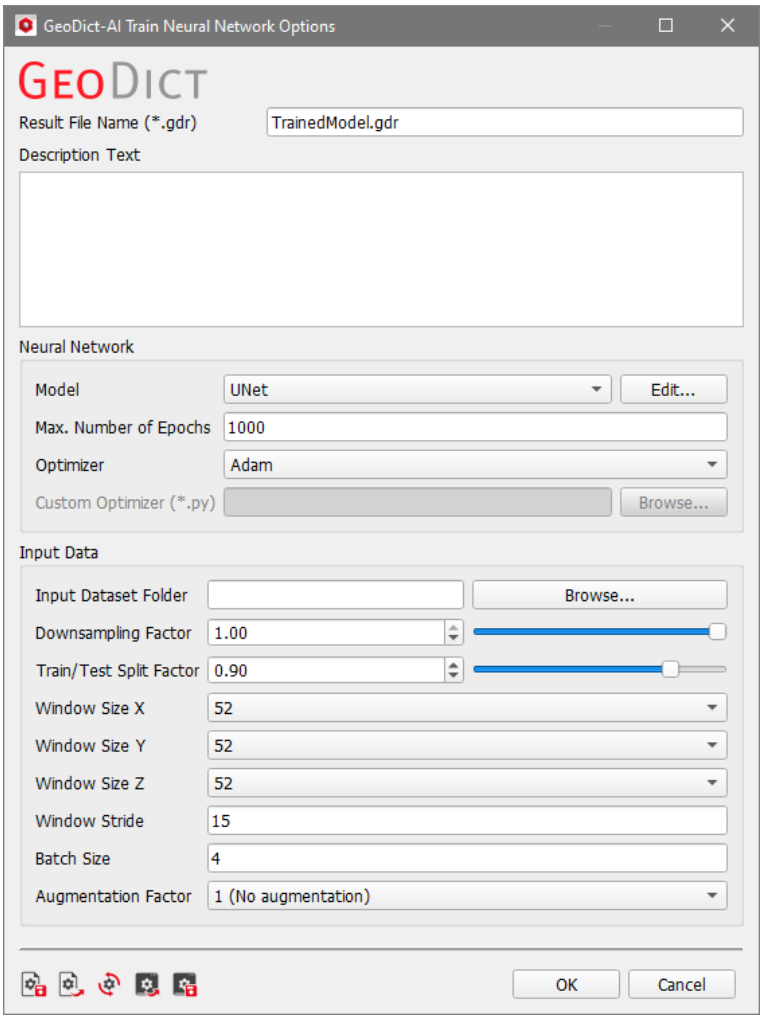
Train Neural Network

If training data is available, a neural network can be trained. In the GeoDict-AI section, select **Train Neural Network** from the pull-down menu. The **GeoDict-AI Train Neural Network Options** dialog opens when clicking the **Options' Edit...** button in the GeoDict-AI section. When opening this dialog the first time, it can take some time as GeoDict checks for available GPUs.

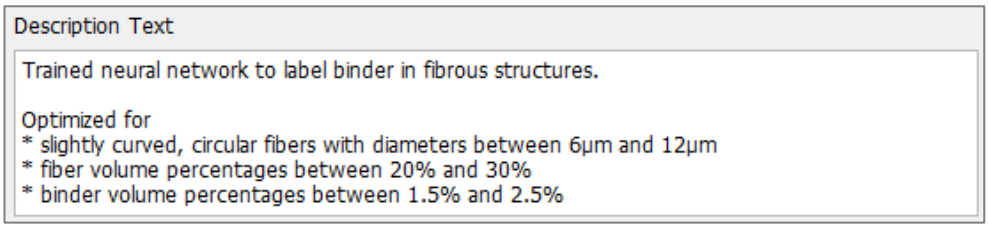


At the top of the **GeoDict-AI Train Neural Network Options** dialog, the name for the files containing the training results can be entered in the **Result File Name (*.gdr)** box. The default name can be kept, or a new name can be chosen fitting the current project.

As usual in GeoDict, the training process produces a .gdr file and a corresponding result folder of the same name. The result file (*.gdr) contains information about the training process. The corresponding result folder contains the trained model.



The **Description Text** can be edited as desired and will be displayed in the **Apply Neural Network Options** dialog when loading the trained network as described starting on page 47. It should contain the purpose of the model and the assumptions made during training, e.g. the fiber diameter range and the solid volume percentage range. Thus, the user applying the model can decide if the model applies to his/her structure.



Note, that the description cannot be changed after the training. Thus, always remember to enter a meaningful description before starting the training.

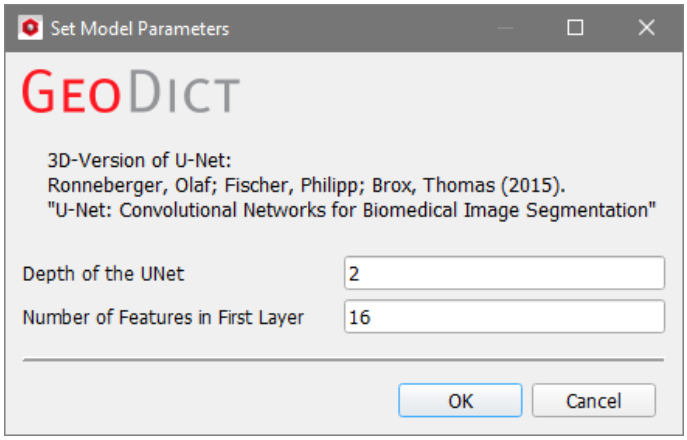
Neural Network

In the **Neural Network** panel define the parameters for the network.

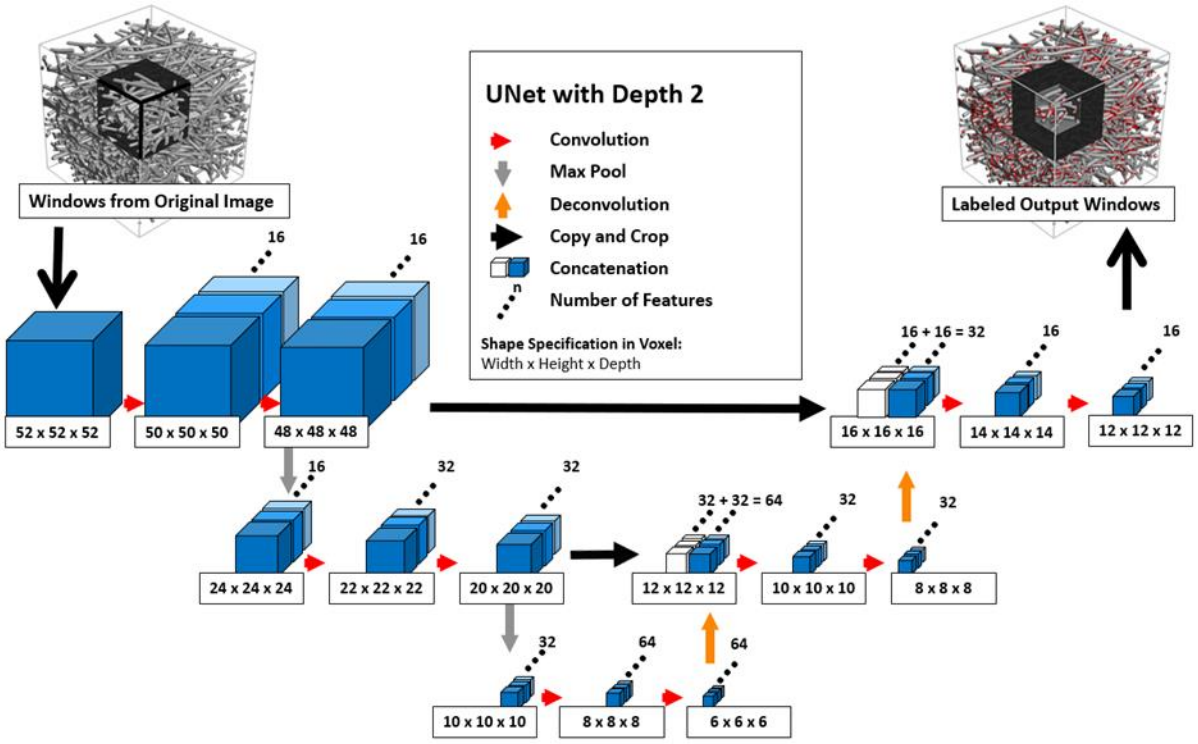
Model

The pull-down menu for **Model** contains the neural network architecture that is being used for training. Currently only the **UNet** is available, which performs very well for segmentation tasks. In future releases, more models will be added. Clicking **Edit** opens the **Set Model Parameters** dialog.

In this dialog the reference [1] describes the underlying architecture of the **UNet**. Increasing the parameters **Depth of the UNet** and **Number of Features in First Layer** improve the performance of the network but also increase the training time and the amount of GPU memory used significantly.



The **UNet** has a U-shape (hence the name) with a constricting and an expanding branch on the left and right respectively. The diagram below shows a UNet with depth 2, 16 features in the first layer and a window size of 52 voxels in X-, Y- and Z-direction. Find a detailed description of the diagram starting on page 6.



The **Depth** of the UNet corresponds to the number of layers in each of the branches. Increasing the depth massively increases the complexity of the network but also the amount of abstraction that it can perform. In [1] the UNet operates in 2D, allowing the authors to use a depth of 4, but since GeoDict-AI performs 3D analyses the limits of GPU memory are reached quickly when increasing this parameter. In general, only 2 or 3 are possible and it is recommended to use the default of 2. In the diagram the depth is the numbers of Max Pool or Deconvolution operations.

The **Number of Features in the First Layer** corresponds to the last shape specification coordinate in the diagram in the first layer on the left, and it determines how many different types of features (edges, corners, blobs...) the network will learn to detect in the first layer. In the diagram, after the first convolution, 16 features are considered. Thus, the number of features in the first layer is 16.

In most cases the default of 16, which is the minimal value, delivers good results for testing the training settings. Once it is ensured, the trained network performs well, larger values can be used for further improving the performance.

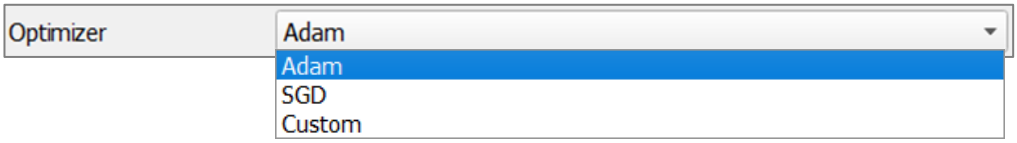
The subsequent layers in the contracting branch will double the number of features in every step. This is a standard construction for these types of networks - in the contracting branch the image resolution is reduced while the information depth (number of features) is increased. Thus, increasing the parameter also increases the amount of GPU memory needed.

Max Number of Epochs

The max Number of Epochs determines how many passes over the data are made during training. The default of 1000 takes a very long time except for the tiniest data sets. In most cases, much smaller values can be used, but as long the needed number of epochs for the current training is not known, run the training with 1000 epochs and observe, when the training converges. Then, the training can be stopped manually, as described on page 40.

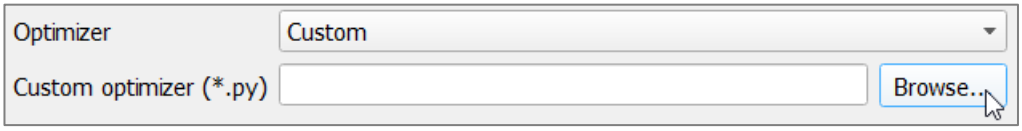
Optimizer

For the **Optimizer** select between **Adam**, **SGD** and **Custom**. The optimization algorithm iteratively updates the network weights (see page 9) according to the current training data.



For many applications, the **Adam** optimizer leads to faster convergence, but the **SGD** (stochastic gradient descent) optimizer can result in a slightly better result. In most cases using **Adam** is recommended. If all other parameters are fitted well to the current training, but the result is not already perfect, run a final production training run with **SGD**. For theory about Adam refer to [2] and for more information about the SGD method see [3].

The **Custom** option allows to specify a Python file containing a definition of a class named **Optimizer**.

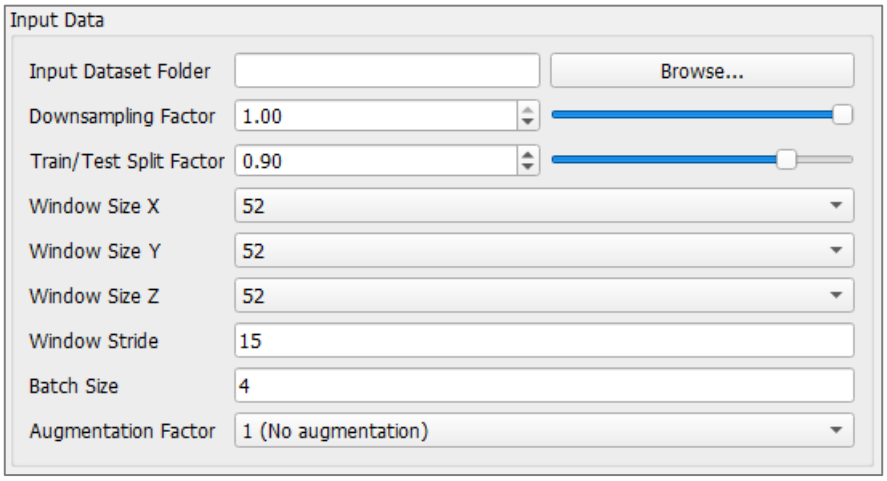


This class must conform to the Tensor Flow optimizer API. To learn more about Tensor Flow refer to the Tensor Flow documentation. When specified, this class will be instantiated and used for optimizing the weights of the neural network.

Note: **Custom Optimizer** is an advanced and experimental feature and for most applications **Adam** or **SGD** will deliver good results.

Input Data

In the **Input Data** panel specify the training data.



Input Dataset Folder

Browse for the **Input Dataset Folder** containing the subfolders **Train** and **Test**. This folder is filled with training data when running **Create Training Data** as described starting on page 25 or manually as described starting on page 16.

Downsampling Factor

The **Downsampling Factor** is a value in [0,1] which can be used to reduce the data amount during training. If this value is smaller than 1, after loading all structures in the dataset, GeoDict-AI will ignore some of the data samples depending on the factor, e.g., if 0.1 is specified, a random 10% subset of the data will be retained. Ensure, that after down sampling, the data is still representative for the experiment design, as it is sampled uniformly from all structures.

For the final training in most cases a down sampling factor of 0.9 or 1 is recommended. However, to test the training, reducing this parameter to 0.05 or 0.1 delivers results in a short time.

Train/Test Split Factor

Before starting with the training, the training structures (images) - generated as explained on page 20 (with Create Training Data) or page 16 (manually) - are split into all the windows defined by **Window Size** (page 34) and **Window Stride** (page 36). After permuting these windows randomly, the **Train/Test Split Factor** determines the number of windows which are used for the training and how many are used for the evaluation during training. This way, the structures (images) from the **Train** folder are divided in train and test data and all training structures are represented in the train data as well as in the test data.

The default value 0.9 means that 90% of the windows are used for training and 10% for evaluation. The 10% of the training data are used for internal tests during the training, i.e. to compute the validation loss (page 38) by applying the neural network to these structures (images) at the end of each sub-epoch. This is necessary to have

a measure on how well the network can generalize what it has learned to the new data. In most cases, the default of 0.9 for the split factor produces good results.

Note, that the structures (images) from the **Test** folder are only used for **Validate Performance** as described on page 58.

Window Size

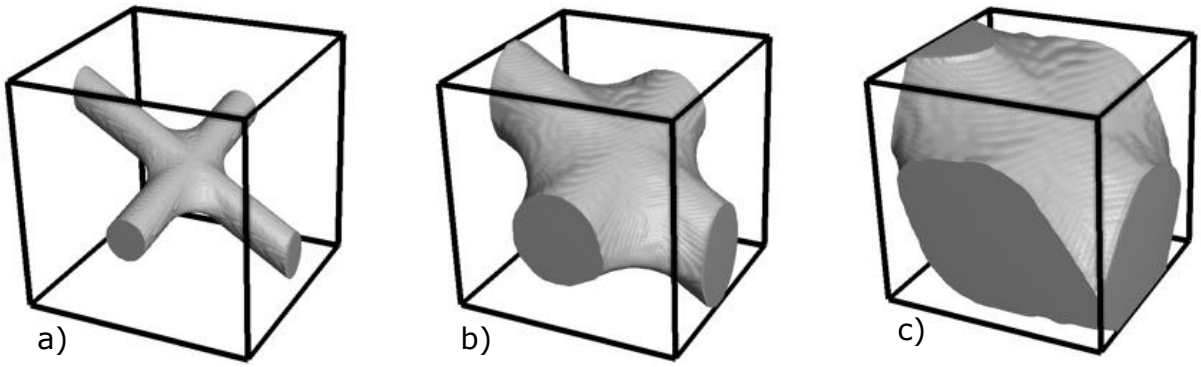
The **Window Size** determines the size (in X, Y and Z) of the sampling window in voxels. When running **Create Training Data** the several input and output files are generated as described on page 27. For the training cutouts from corresponding **input** and **output** files are taken in the same location with the given window size. Thus, all network's decisions are based on these cropped data pairs alone. The neural network will try and learn to transform what it sees in the input window to what it observes in the output window. The **Window Size** determines how much context the network obtains for its decisions. Larger values improve the performance, while increasing training time and GPU (memory usage).

The **Window Size** must be large enough to capture an entire important feature, e.g. a fiber crossing with binder. This way it is possible to distinguish the different materials only from the window.

For a fiber structure, for example, it is recommended to choose the window bigger than twice as large as the diameter of the largest fiber. This allows two touching fibers to be fully contained in a single window and therefore the network can correctly identify the geometric configuration, as two fibers crossing or touching and hence, make an informed decision about the binder placement around the touching point.

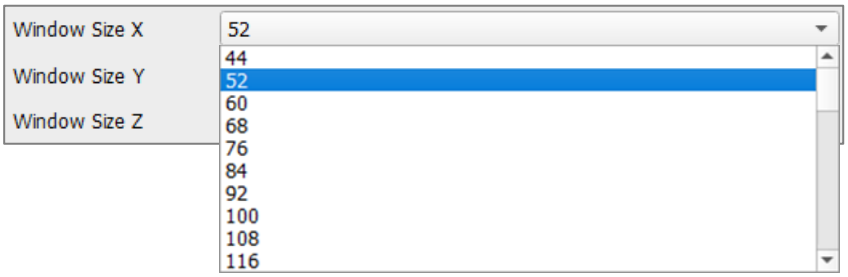
Consider differentiating binder and fibers only based on a cutout in the selected window size. If a human cannot distinguish the material phase from the geometry seen in the cutout window, it is unlikely that a neural network is capable to. In such cases, the window size must be increased so the neural network can "see" the two fibers approaching each other and trace them through the intersection even with the large amount of binder obscuring the actual intersection point. If it is humanly possible to identify the binder, the network can also learn that. Otherwise, the window size must be increased.

In the next figure, different window sizes are considered. Observe how in a) it is easy to distinguish binder from fibers, as the window size is larger than twice the fiber diameter. In image b) it is already hard, but still possible with a window size only a little more than twice the fiber diameter. However, in a cutout even smaller than twice the fiber diameter as in c) it will be nearly impossible to identify the fibers.

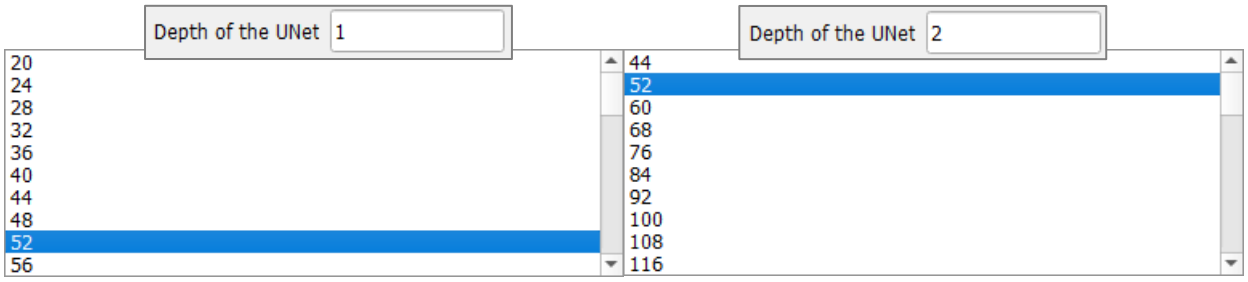


The aspect ratio of the window can be adjusted to the corresponding structure, e.g. for a flat textile structure or for materials which are anisotropic. For example, in the gas-diffusion-layer of a fuel-cell the fibers are usually oriented mainly in the X/Y-plane. Then, a smaller window size in Z-direction is reasonable. For isotropic materials, however, it is recommended to set the window size for all three directions to the same value.

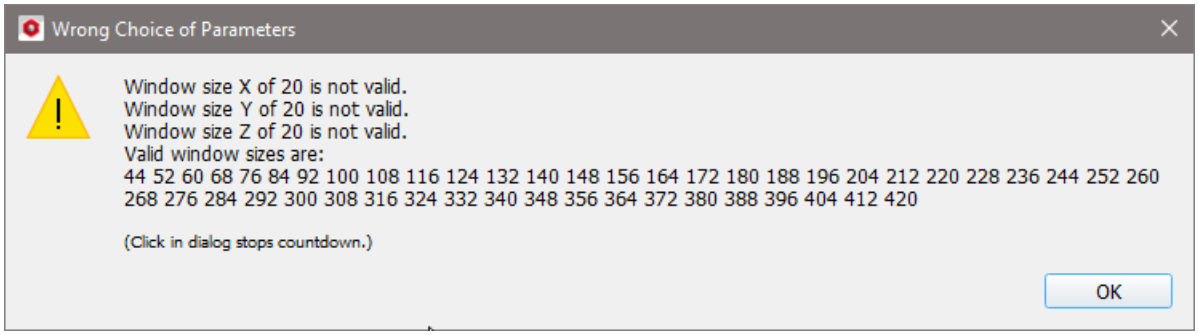
The possible values for **Window Size** also depend on the **Depth** selected for the **UNet** model. In each convolution 1 voxel is lost at each border of the cutout. As there are two convolutions in each layer of the UNet, four voxels are subtracted from the window size in each direction (compare to the UNet diagram above). The resulting values are divided by two in each max-pool operation. The selected **Depth** determines the number of applied max-pool operations. How to edit the **Depth** is described on page 31.



Select the **Window Size** from the pull-down menu. After changing the **Depth** in the **UNet** dialog, click **OK** to close the **Train Neural Network Options** dialog. Reopen it by clicking **Edit** in the **GeoDict-AI** section to update the pull-down menu for window size.



If wrong window sizes are chosen an error message appears when closing the **Train Neural Network Options** dialog, displaying possible values for the selected depth.



In the following table find possible window sizes for reasonable depth values. While the pull-down menus also offer larger values, in most cases a window size larger than 200 will not be possible with the given hardware.

Depth	Possible Window Sizes
1	20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 144, 148, 152, 156, 160, 164, 168, 172, 176, 180, 184, 188, 192, 196, 200, 204,...
2	44, 52, 60, 68, 76, 84, 92, 100, 108, 116, 124, 132, 140, 148, 156, 164, 172, 180, 188, 196, 204, ...
3	92, 108, 124, 140, 156, 172, 188, 204, …
4	188, 220, 252, 284, 316, 348, ...

Window Stride

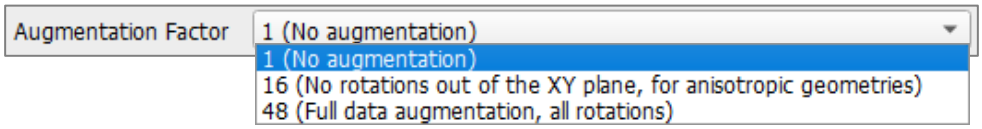
The **Window Stride** determines where the windows defined by the window size are placed in the structure. Each location of a window delivers a cutout used as one training sample. Starting at the upper left corner, the window is sliced through the structure. A **Window Stride** of 1 means, that every possible window position in the structure is considered. This is not recommended in most cases, as shifting the window by only one voxel results in too many similar cutouts, taking too much time, while not gaining new information. The default value of 15 means, that the window is shifted 15 voxels for each new sample. Thus, a higher number of different situations can be sampled for a given time/memory budget. Increasing the **Window Stride** has a similar effect as decreasing the **Down Sampling Factor** – leading to less samples. While for the final training run it can be necessary to reduce the window stride, for testing higher values are recommended.

Batch Size

The **Batch Size** is the number of samples the training uses simultaneously. For example, with the default batch size 4, during training the error is computed, and the network updated for four windows at the same time, before moving on to the next four windows. A value of 1 slightly decreases training efficiency and can result in a noisier training. For example, if a window contains an uncommon geometric configuration, the training could take a step into the wrong direction when computing the updated network weights for the next step. In literature there is usually a warning about too large values for batch size due to the potential of overfitting, but in 3D image analysis here the batch size is limited by the available GPU memory. If the value is chosen to high, a warning dialog appears, when starting the training. It is recommended, to set the value as high as possible with the available GPU memory, so that the training works without giving an error. This is often a value between 1 and 8 depending on the GPU hardware.

Augmentation Factor

The **Augmentation Factor** allows to effectively increase the amount of training data.

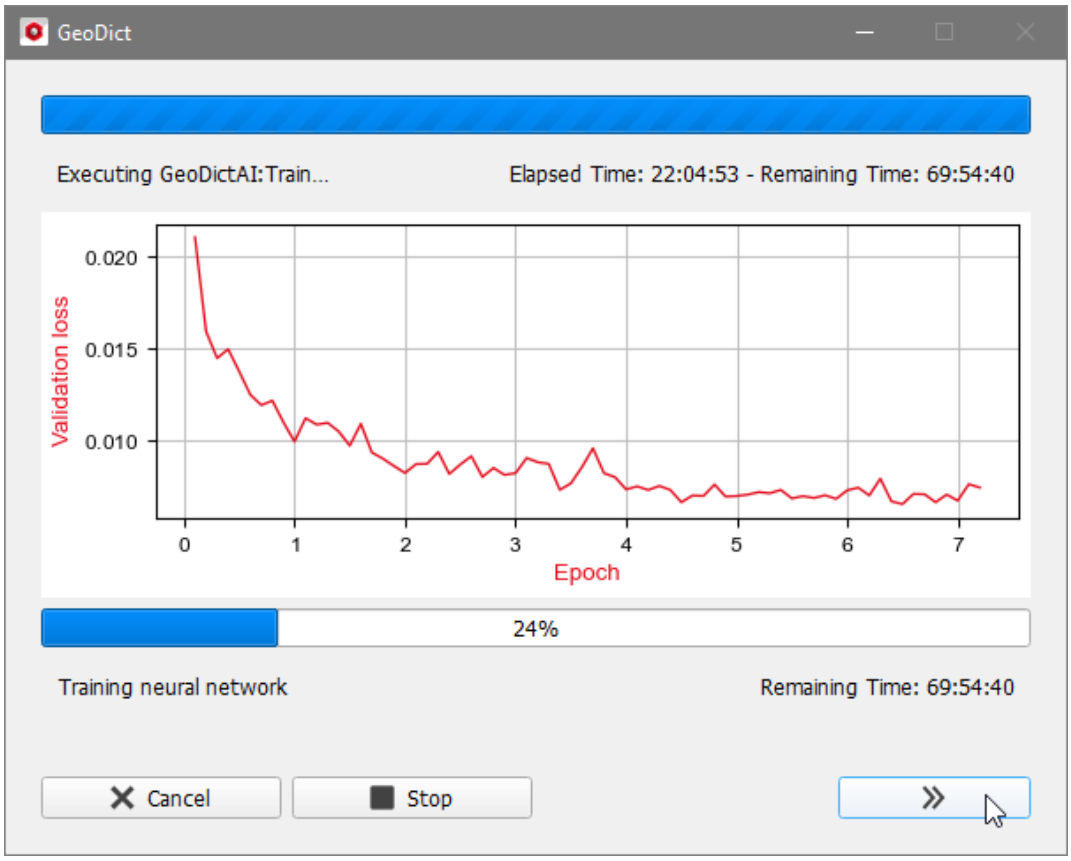


For the default of **1 (No Augmentation)**, each window is seen once during a training epoch. Setting it to **48 (Full data augmentation, all rotations)**, the window will also be mirrored and rotated, effectively increasing the dataset size by factor 48 with the corresponding increase in runtime. The option of factor **16 (No rotations out of the XY plane, for anisotropic geometries)** which is specifically for structures where fibers are mainly oriented in the XY-plane. Then, only mirroring and rotation in the XY-Plane is performed. However, values higher than 1 are mostly useful for a small dataset, e.g., a single structure, to get as much information as possible from it.

Run Training

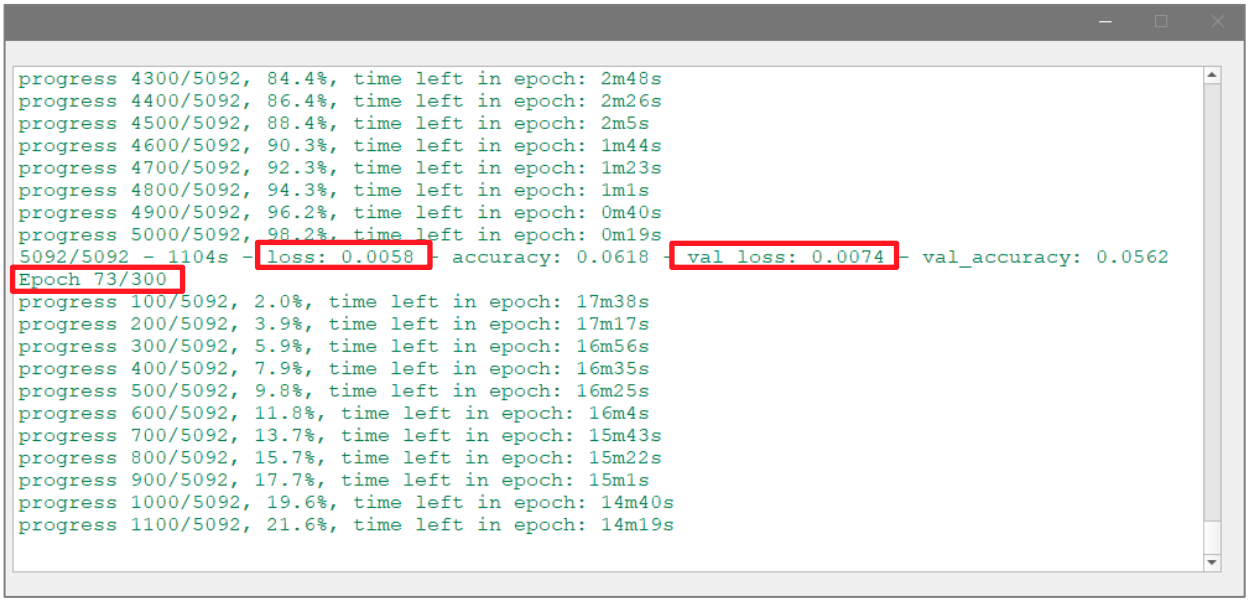
After setting the parameters click **OK** to close the dialog.

After clicking **Run** observe the training in the progress dialog and for more details open the **GeoDict Console** by clicking on the double arrow in the bottom right of the dialog.



Epoch

As an epoch is a single pass over all the data and thus, often takes much time, each epoch is internally subdivided into ten sub epochs. This leads to more frequent intermediate results. After each sub epoch, the current training and validation loss figures are shown in the console and a .gnn file is written to the result folder for every step. Currently, in the console the sub epochs are counted as epochs. Thus, setting the **Max. Number of Epochs** in the **Train Neural Network Options** dialog to 30 leads to 300 epochs in the console.



Training Loss and Validation Loss

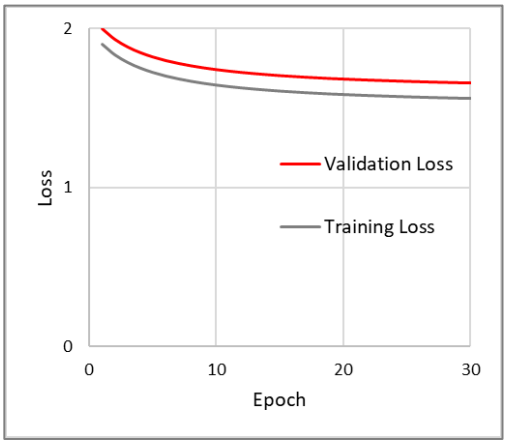
To consider the convergence, during training observe two numbers in the **GeoDict Console**: **Training Loss** (**loss**) and **Validation Loss** (**val_loss**). These correspond to the two subsets of the data defined by the **Train/Test Split Factor** set up in the **Train Neural Network Options** dialog. In neural network training the term "Loss" defines the current error of the neural network. The **Training Loss** is computed during each sub epoch while training on the training subset of data and the **Validation Loss** is computed while applying the current network of the sub epoch on the validation subset of data.

In the progress plot on the left the convergence behavior is visualized for the **Validation Loss**.

Ideally, both loss values converge towards zero, meaning zero error. In the following the three different scenarios that can occur in training are discussed:

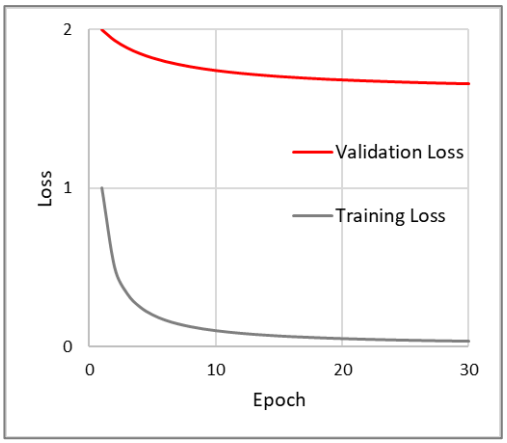
1. **Both losses do not go down/stagnate:**

   Either the network does not have enough context in a window, and the **Window Size** must be increased, or the nature of the problem is too complex for the network and the **Depth** or the **Number of Features in First Layer** must be increased.



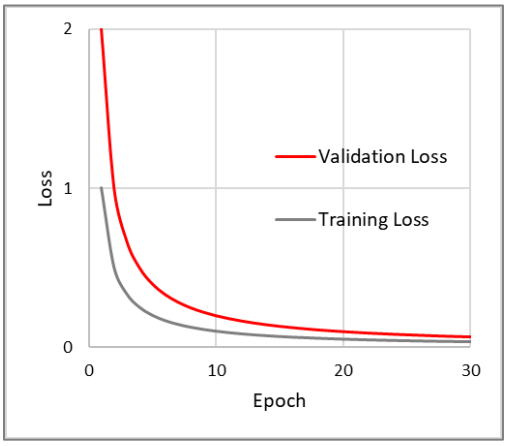
2. **Training loss goes down, validation loss does not**:

   This is usually a sign of overfitting, e.g., the network "memorizes" the training data and does not generalize to the test data. The solution is usually to feed in more data, e.g., by generating more **Training Structures** or **Training Images**.



3. **Both losses go down and converge towards zero:**

   The training converges and thus, the network works well on the training data and generalizes to the test data.



As the scale of the figures vary with the parameters in structure generation, it is currently not possible to specify a "good" numerical value for the loss. Consider the shape of the convergence for the current application and remember the values for similar trainings in the future to estimate the training performance. As described in the categorization above, the **Validation Loss** is the more important number to consider primarily.

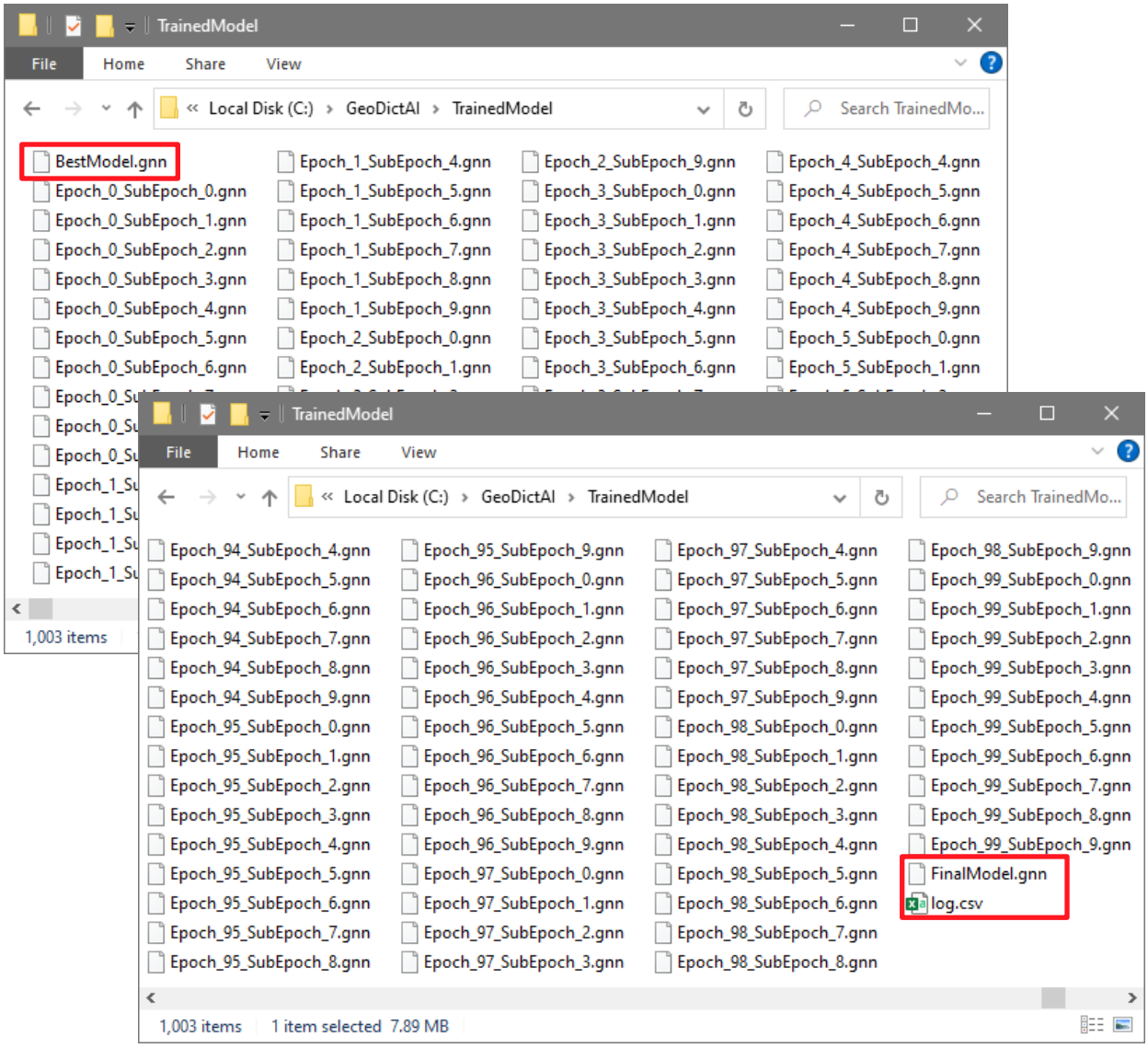
Cancel or Stop Training

The training is run until the **Maximal Number of Epochs** is reached or the training is stopped manually. It is recommended to click **Stop** rather than **Cancel**, as **Stop** still produces a result file and keeps all results already generated and saved in the result folder, while Cancel deletes the complete result folder. It can be necessary to stop the training manually, if for example a too large number of epochs was selected, and the training already converges long before the number is reached.

RESULTS

The training produces a GeoDict result file (*.gdr) and a folder with the same name. Both are saved in the chosen project folder (**File → Choose Project Folder...** in the menu bar). Often the latest neural network performs best, but if the training convergence is not monotonous, this can be different. Thus, the result folder contains a **GeoDict Neural Network** (*.**gnn**) for each sub epoch and two result networks:

- **FinalModel.gnn:** The final model is the last neural network with the largest epoch number. Thus, in the figure below epoch_99_subepoch_9.gnn and FinalModel.gnn are the same neural networks.
- **BestModel.gnn:** The best model is the neural network with the minimal validation loss. Find the corresponding epoch and sub epoch in the Result Viewer as shown below. During the training in every epoch, when a new minimal loss is found, the corresponding *.gnn file is copied and renamed to BestModel.gnn.



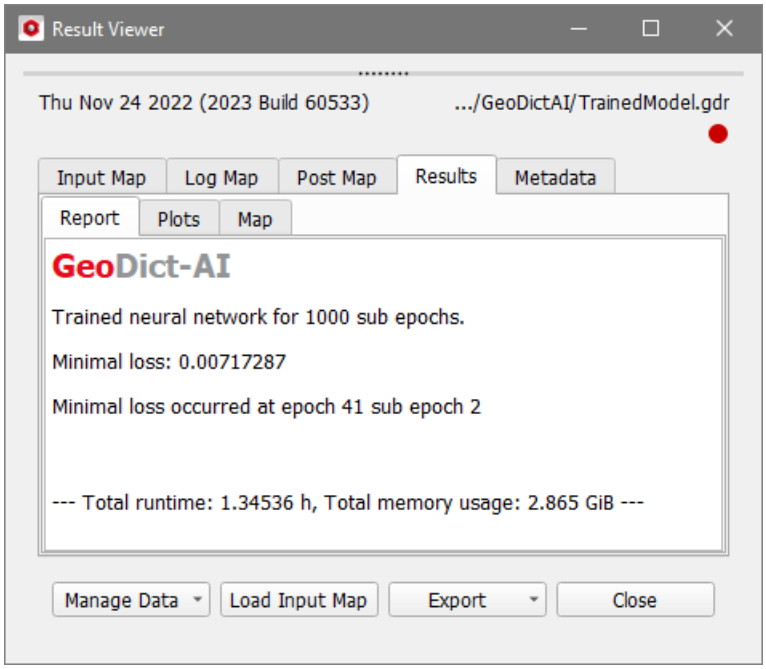
Additionally, find the **log.csv** file in the result folder updating with each sub epoch and containing a table with all training and validation loss values for each sub epoch. The file can be opened in Excel and many other text editors.

	A	B	C	D	E
1	epoch	accuracy	loss	val_accuracy	val_loss
2	0	0	0.407690257	0	0.712476075
3	1	0	0.302275032	0	1.497899771
4	2	0.013888889	0.24575159	0	1.0922153
5	3	0.027777778	0.213734195	0	0.487241477
6	4	0.027777778	0.175469369	0.125	0.867811799
7	5	0.027777778	0.160106152	0.125	0.613530874
8	6	0	0.138903722	0	0.281490624
9	7	0.055555556	0.124918193	0	0.21729672
10	8	0.069444448	0.109046429	0.125	0.196926773
11	9	0.083333336	0.111665405	0.125	0.13103646
12	10	0.055555556	0.098318778	0	0.179819256
13	11	0.083333336	0.098529682	0.125	0.150994927
14	12	0.069444448	0.090456478	0	0.1221634
15	13	0.041666668	0.093718186	0	0.114446878
16	14	0.027777778	0.087733299	0.125	0.09532465
17	15	0.055555556	0.06983088	0.125	0.112935975
18	16	0.055555556	0.07539665	0	0.091588475
19	17	0	0.080670655	0	0.084429309
20	18	0.041666668	0.077344388	0	0.124674864

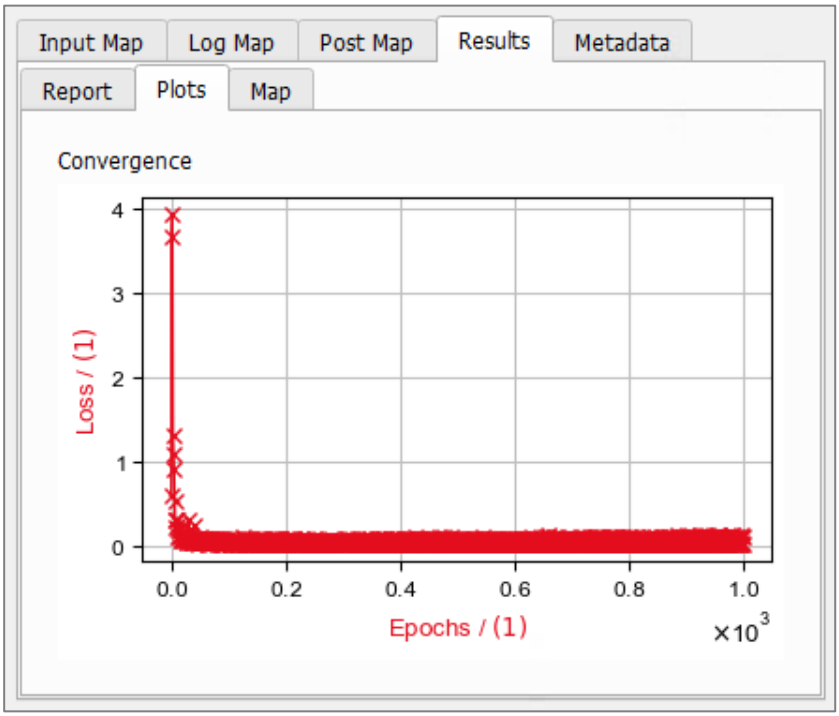
The GeoDict **Result Viewer** opens for the result file and statistical properties of the training are shown in the **Results – Report** subtab.

The **Number of sub epochs** used for training gives information about how often the training data was passed.

The **Minimal Loss** is the smallest **Validation Loss**, that occurred during the training. The sub epoch corresponding to this loss value is given in the last line of the report.



In the Result Viewer, under the **Results - Plots** subtab, find the **Convergence** plot showing the **Validation Loss** for each **Epoch**.



As soon the training is finished, the resulting networks **BestModel.gnn** and **FinalModel.gnn** can be validated with **Validate Performance** described starting on pages 58. If the networks are trained to distinguish between material phases, they can be applied with the **Apply Neural Network** option described in the next chapter.

If networks are trained to enhance 3D-scans, they can be applied with **Enhance Grayscale Image** as described on page 54.

Otherwise, if networks are trained to identify individual fibers, use FiberFind **Identify Fibers (AI)** as described in the FiberFind handbook.

Apply Neural Network

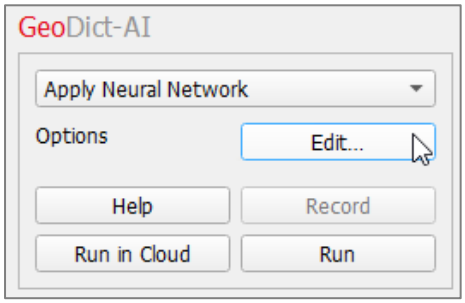
After producing training data and training neural networks, the networks can be applied to all structures with statistical parameters fitting to the network.

Use GeoDict-AI **Apply Neural Network**, to apply neural networks trained to distinguish between material phases.

However, if a network is trained to identify fibers, the FiberFind **Identify Fibers (AI)** module must be used. Refer to the FiberFind handbook for more information.

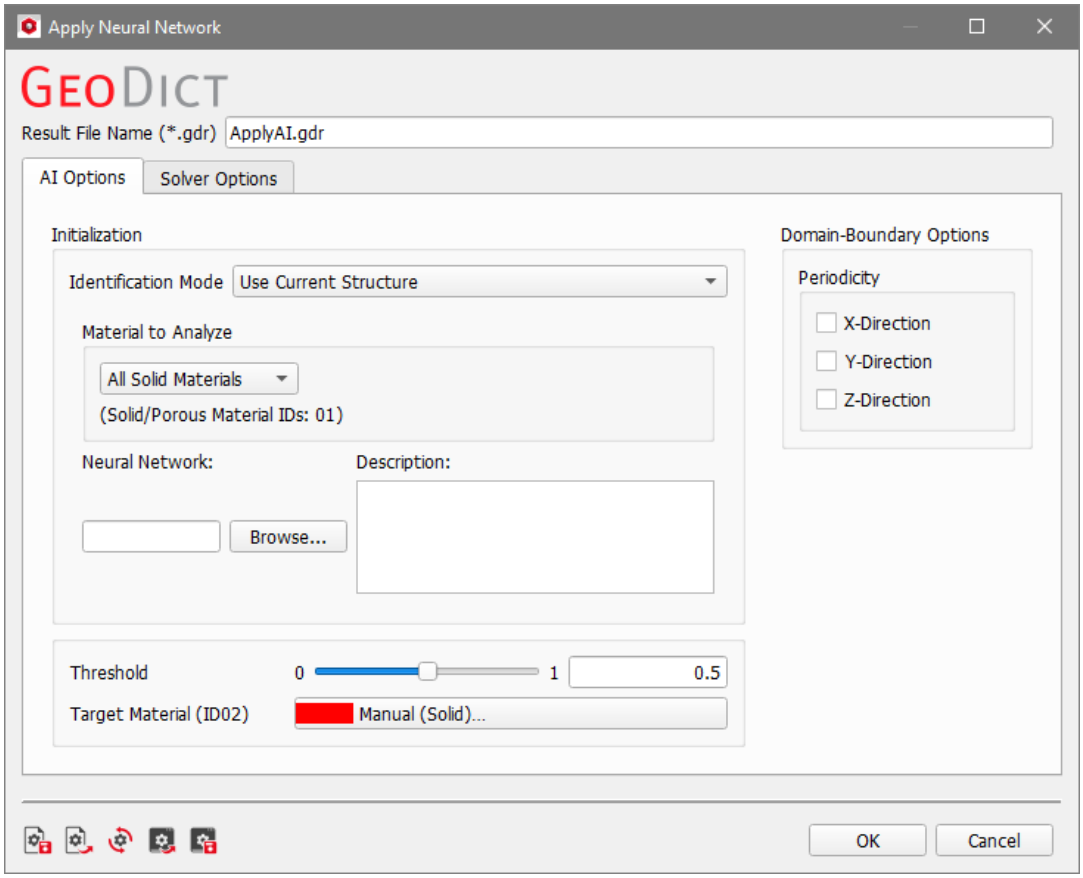
If a network is trained to enhance images use **Enhance Grayscale Image** as described starting on page 54.

To start, load the structure to analyze and select **Apply Neural Network** from the pull-down menu in the GeoDict-AI section.



Click the **Options' Edit...** button. When opening this dialog the first time, it can take some time as GeoDict checks for available GPUs.

The **Apply Neural Network** dialog is organized in two tabs: **AI Options** and **Solver Options**.

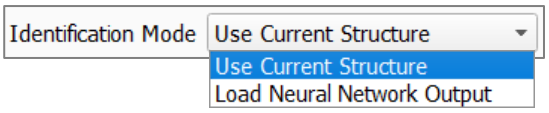


AI-Options

Initialization

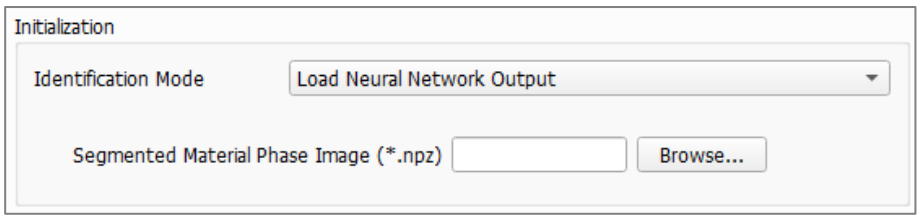
Identification Mode

Two choices are available as **Identification Mode**: **Use Current Structure** and **Load Neural Network Output**.



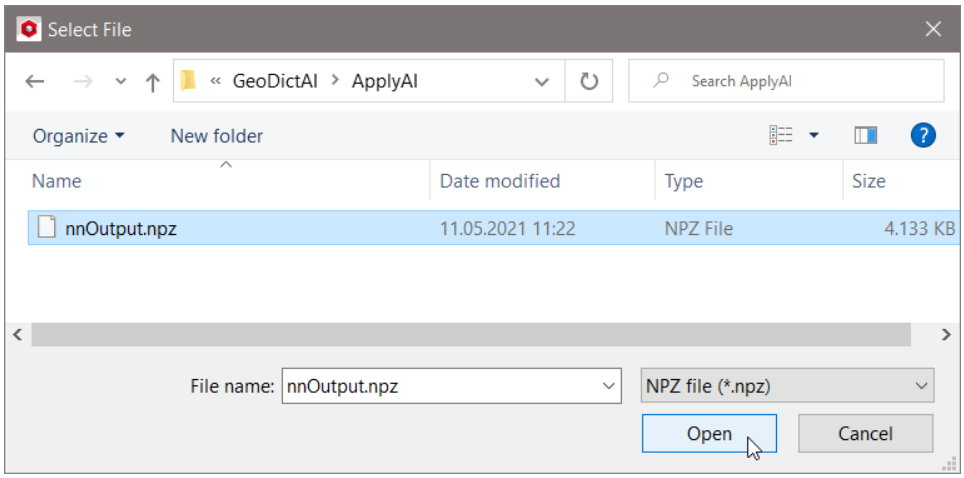
The default **Use Current Structure** applies GeoDict-AI to the structure currently in memory.

In occasion, the option **Load Neural Network Output** may be very useful, if different thresholds should be tested, without having to run the identification for each voxel with the neural network again. Another application would be, that the identification was not finished but an intermediate *.npz file is saved. Then this file can be loaded and thresholded to find out, if the identification is already good.



How to define the threshold is described on page 47 and a visualization of the confidence field and the threshold is shown on page 52.

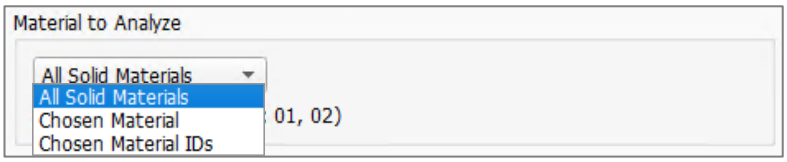
Browse to a GeoDict result folder of a previous run of GeoDict-AI **Apply Neural Network** and select a nnOutput.npz file from that folder. Accordingly, the **Material to Analyze** and the **Neural Network** to be used cannot be changed, and the loaded structure needs to be the same.



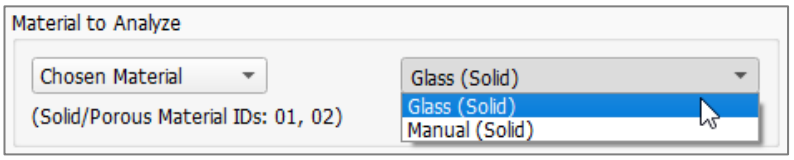
However, for most application cases, the default values can be kept, as the default threshold of 0.5 usually delivers good results for a well-trained network.

Material to Analyze

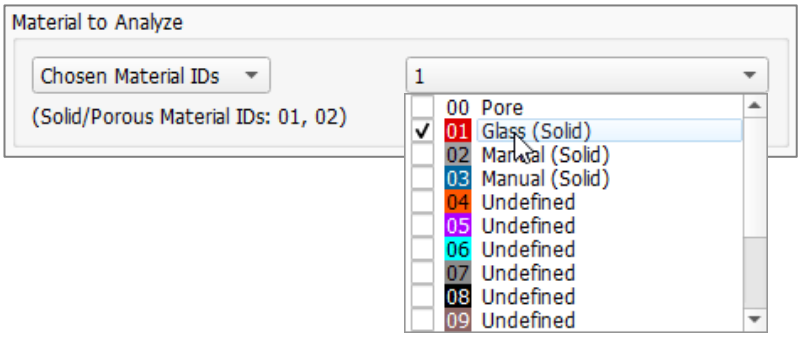
The **Material to Analyze** panel offers the choice of whether GeoDict-AI should be applied to all solids in the structure or only on a subset, defined either by choosing material IDs or materials.



If **Chosen Material** is selected, select the material to analyze from the pull-down menu on the right. All other materials will be considered as background by the neural network.

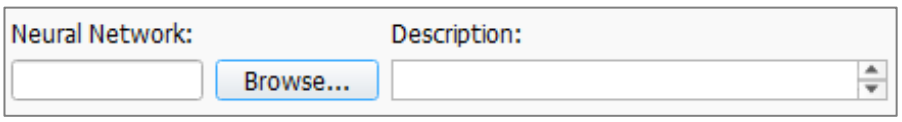


If **Chosen Material IDs** is selected, choose the material IDs to analyze from the pull-down menu on the right. All other material IDs will be considered as background by the neural network.

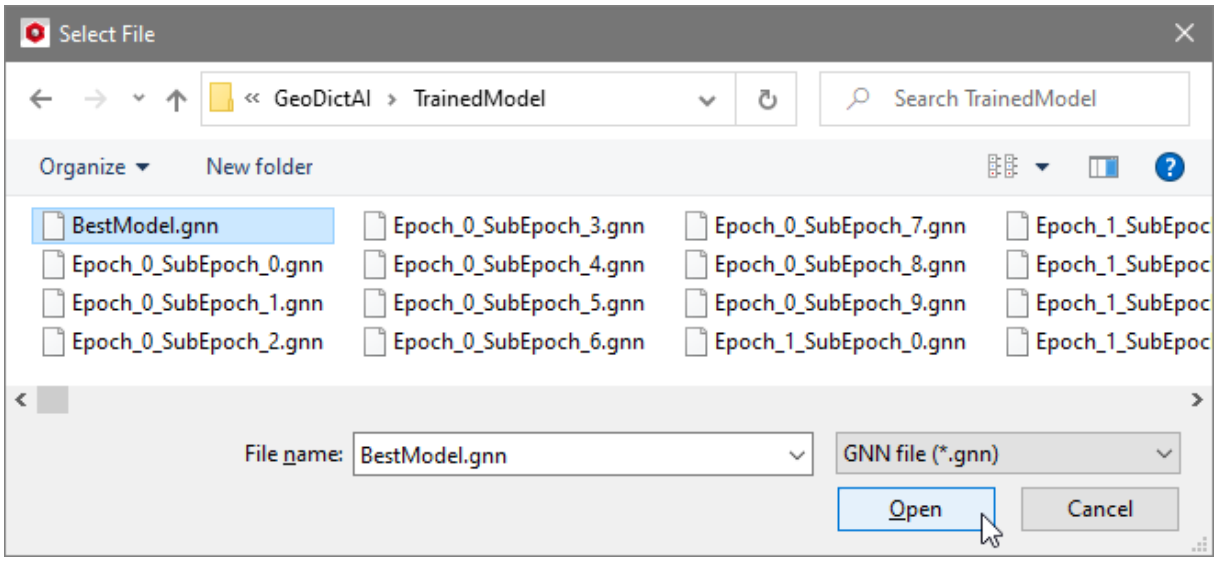


Neural Network and Description

Browse for the neural network trained with **Train Neural Network** and best suitable for the structure under consideration. Therefore, under **Neural Network** click **Browse**.



Select **BestModel.gnn** or **FinalModel.gnn**. The difference between these two models is described on page 41, but usually using **BestModel.gnn** is recommended.



In the **Description**, the current constraints for the application of the neural network are listed, as entered in the **Train Neural Network** dialog described on page 30, e.g. the diameter range of the fibers the neural network was trained for.

Threshold

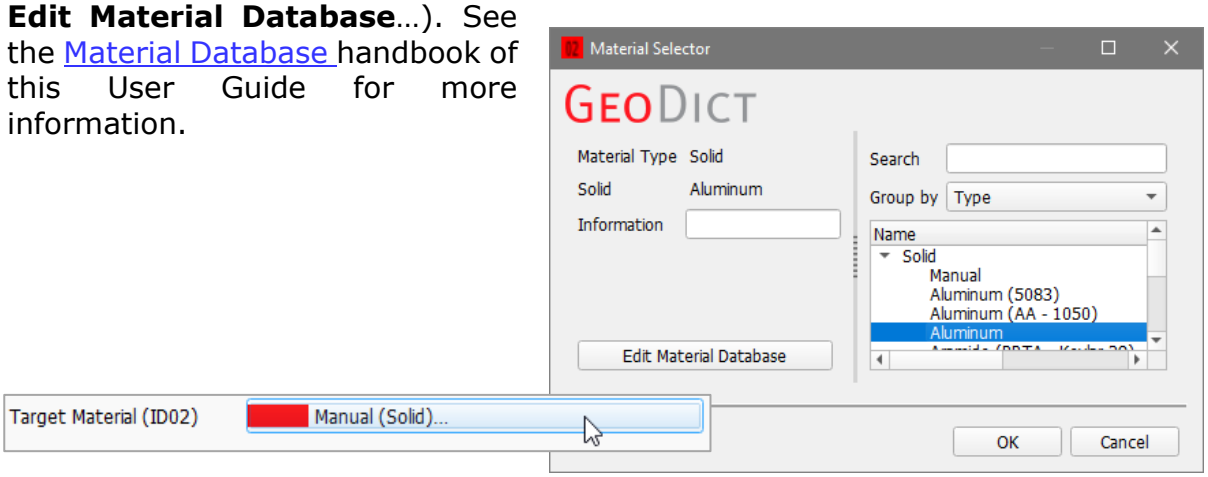
The **Threshold** is an internal (or expert) parameter that is mostly applied by proficient users at Math2Market. It is used for the decision whether a voxel is a target material voxel or not. If the identified features are over segmented, choosing a smaller threshold can lead to better results. Therefore, the confidence field (*.npz) can be loaded again as described on page 45, without running the complete identification process again.

Find a visualization of the confidence field (*.npz) and threshold on page 52.

Target Material

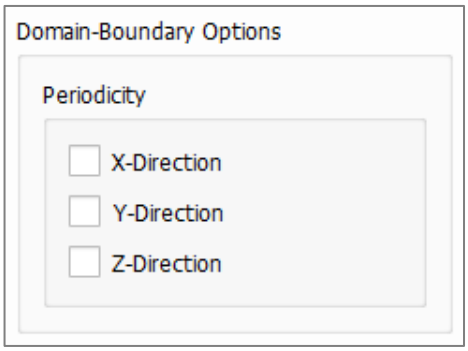
Target Material sets the material assigned to the target material, e.g. the binder material. The **Material Selector** gives access to selecting the desired material from the GeoDict Material Database. When none of the materials available in the database fits the preferred specifications, **Manual** should be chosen.

Alternatively, new materials can be defined in GeoDict's material database (Click **Edit Material Database**…). See the Material Database handbook of this User Guide for more information.



Domain Boundary Options

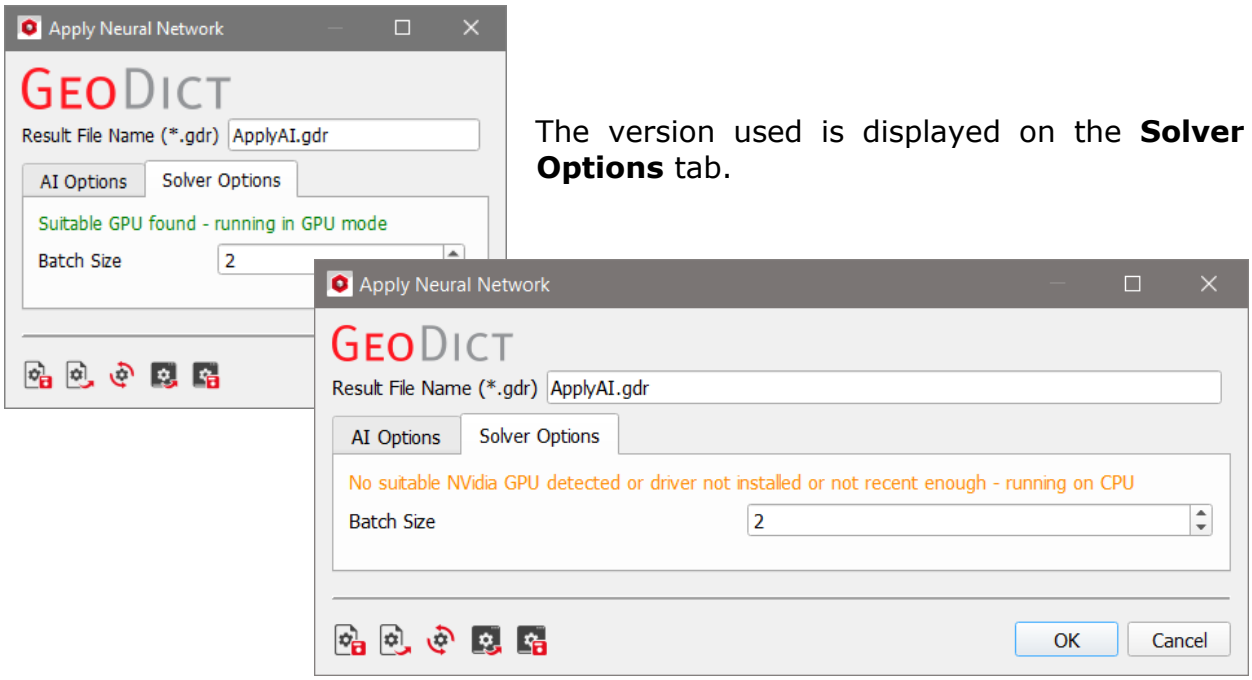
The underlying domain can bet set as periodic for the object identification by checking one or several of the boxes in the **Domain-Boundary Options** panel.



However, periodicity is unusual for most 3D scans and this setting is rarely changed.

Solver Options

If a suitable graphics card is detected during installation of GeoDict, the GPU mode is used for running GeoDict-AI, otherwise it is running in CPU mode.

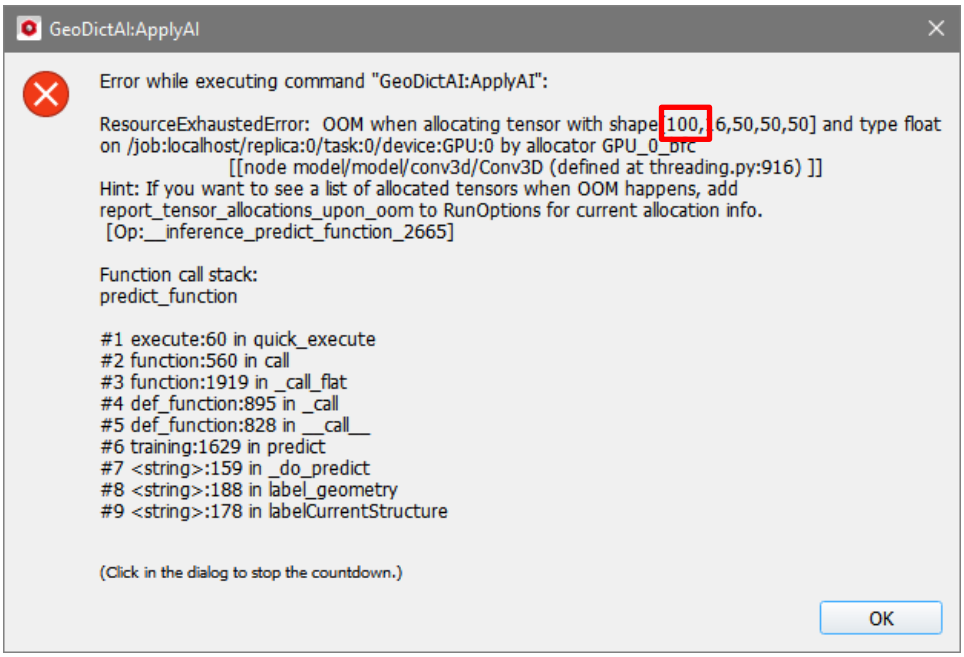


The version used is displayed on the **Solver Options** tab.

Batch Size

The choice for **Batch Size** is related to the memory available on the graphics card (GPU). Conceptually, GeoDict loads the graphics card with portions of work called batches. Currently, the selection must be made manually, and the parameter is set following the value entered in **Batch Size**.

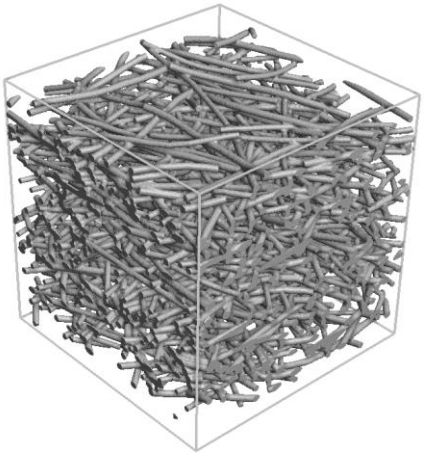
The batch size might be chosen too large for what is available on the graphics card. In this case, an error message appears. Here, a batch size of 100 was used, which is much too high.



In this case, the number of batches needs to be reduced. Note that for CPU-based computation, the choice of batch size is not critical and can be left at the default.

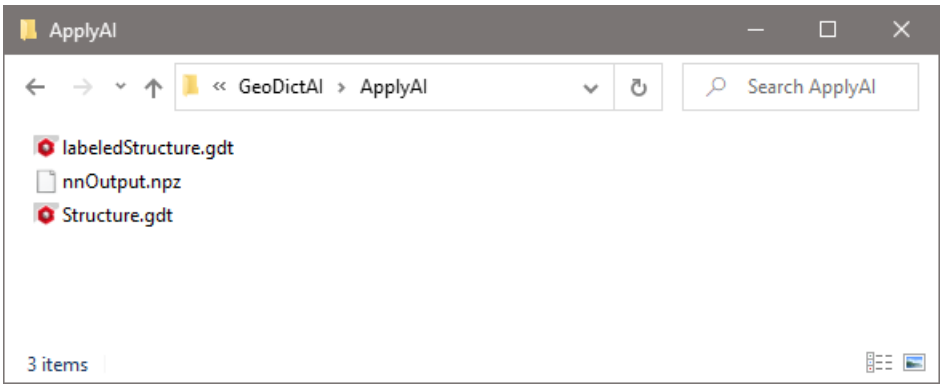
After setting the parameters as desired, close the dialog by clicking **OK** and click **Run** in the GeoDict-AI section.

Results

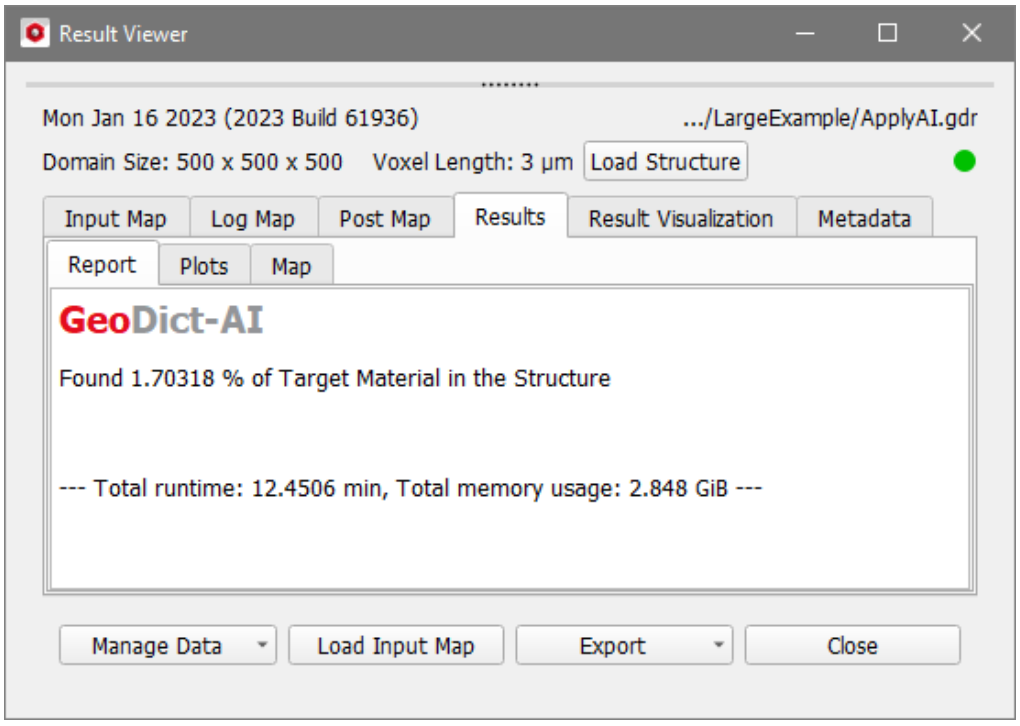


After applying the neural network, a folder with the same name as the GeoDict result file is saved in the chosen project folder (**File → Choose Project Folder...** in the menu bar).

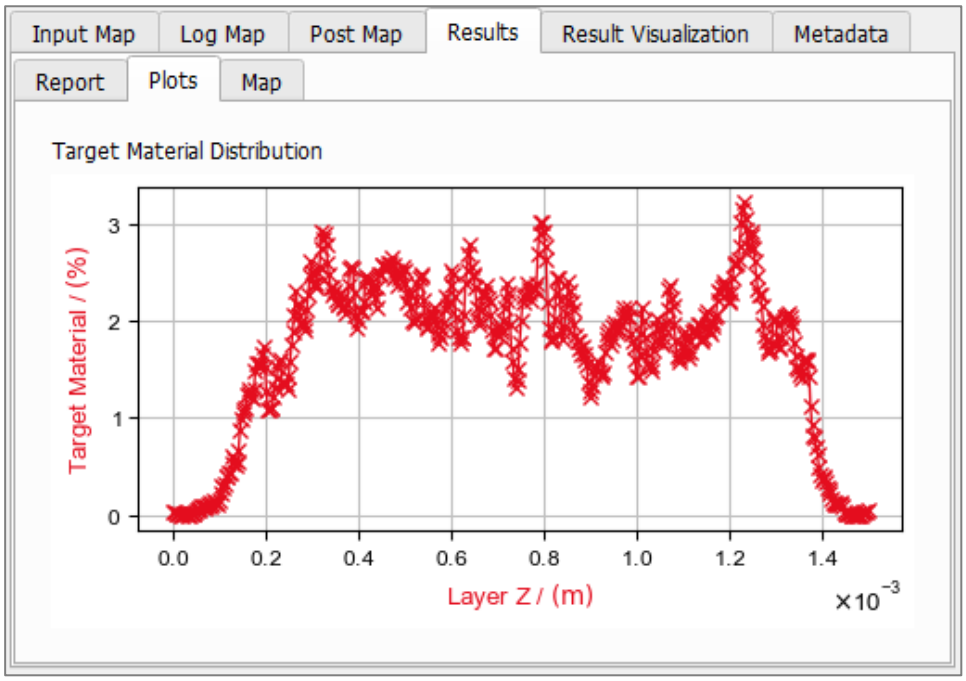
This folder contains the original structure model on which the identification was done (**Structure.gdt**) and the resulting structure with identified target material (**labeledStructure.gdt**). Also, the neural network output file (**nnOutput.npz**) is saved in the folder and can be reused as described on page 45.



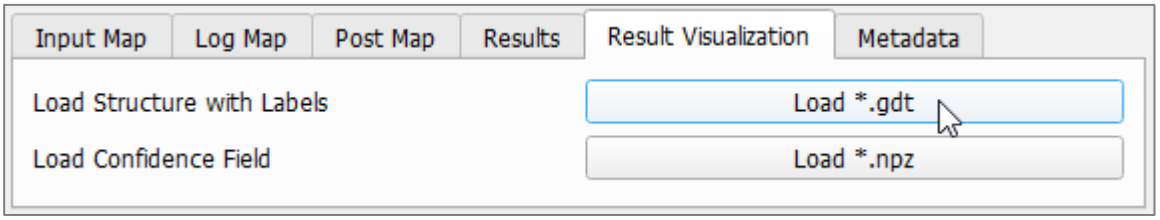
The GeoDict **Result Viewer** opens for the result file and in the **Report** the solid volume percentage of the labeled target material is given.



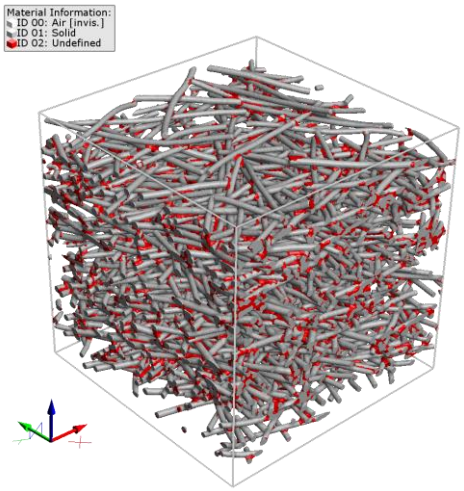
In the **Result Viewer**, under the **Results - Plots** subtab, the find the solid volume percentage distribution of the target material per slice in Z-direction.



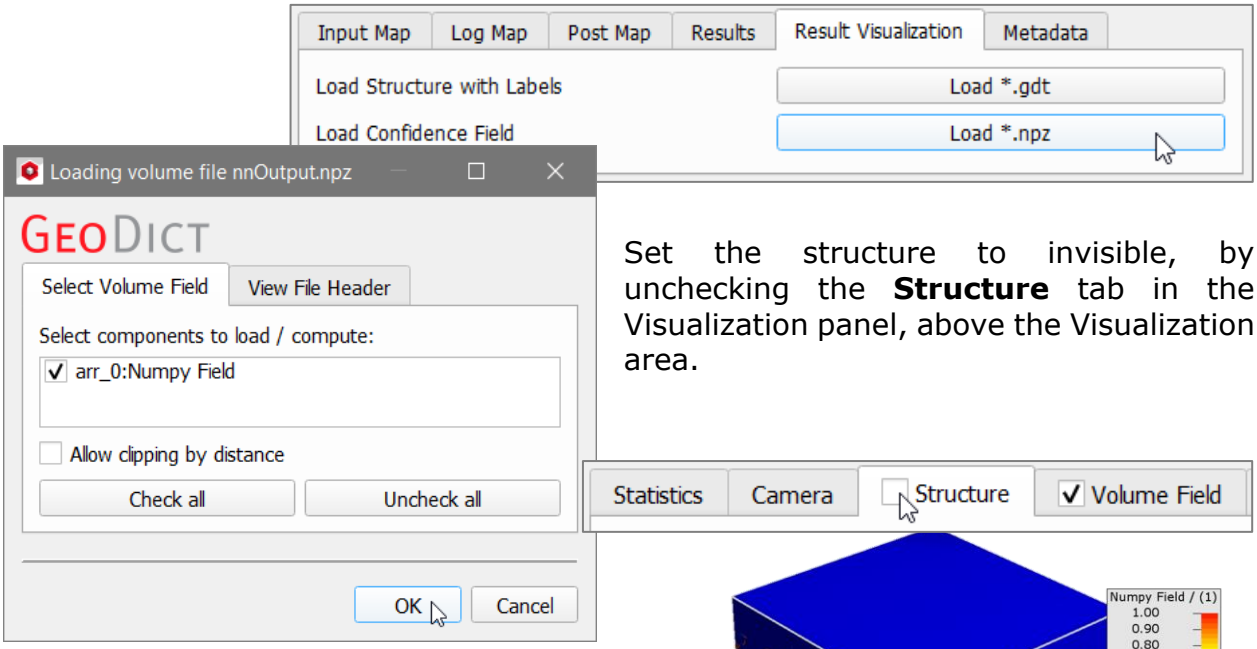
To visualize the target material identified by the network move to the **Result Visualization** tab. There, click **Load *.gdt** for **Load Structure with Labels**.



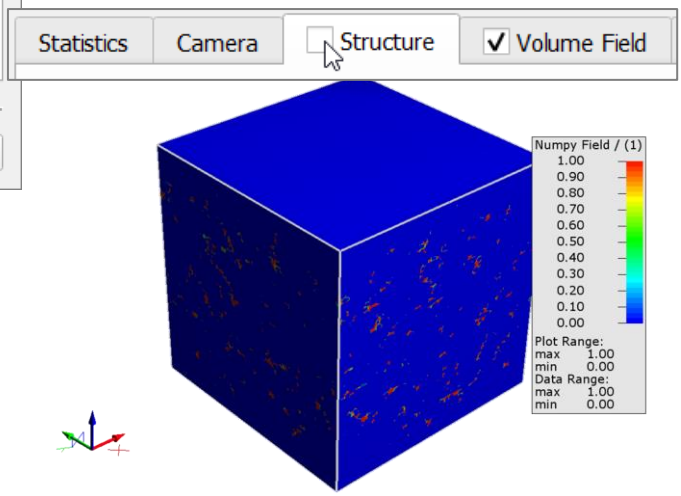
The identified target material is assigned to Material ID 02 and thus, is shown in the color selected for this ID.



Click the **Load *.npz** button next to **Load Confidence Field** to load the unassigned data of the material identification.

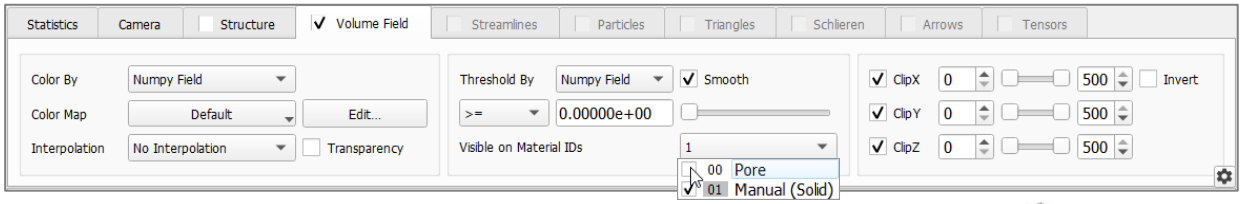


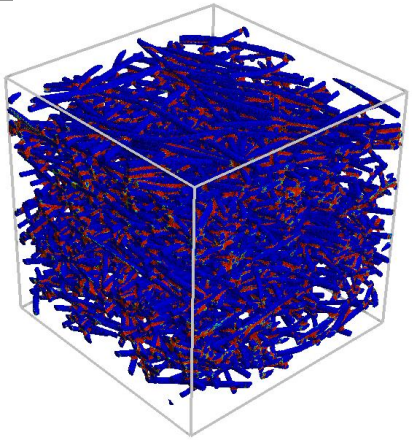
Set the structure to invisible, by unchecking the **Structure** tab in the Visualization panel, above the Visualization area.



Move to the **Volume Field** tab to control the visualization of the confidence field.

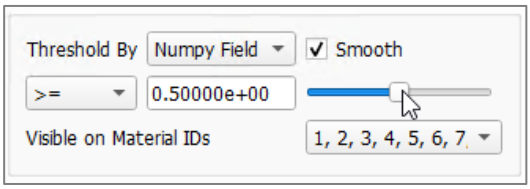
For example, to only see the values of interest deselect ID 00 for **Visible on Material IDs**.

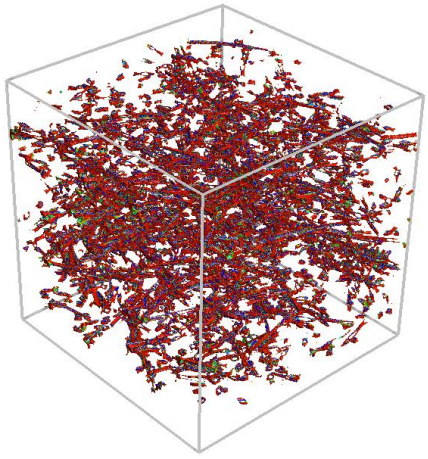




Also, the **Threshold** for the **Target Material** can be visualized. Therefore, change the clipping value to separate the values for the different materials. If the **Threshold** was set to **0.5** as explained on page 47, entering 0.5 for the volume field threshold and selecting **>=** as clip mode visualizes only the result values for the identified target material. For a well-performing network most of the values in the volume

field should be near 1 (target material) or near 0 (original material), meaning the network is confident with the classification for the corresponding voxel. This can be obsered in the example below. After clipping away all voxels with a value smaller than 0.5, most of the remaining voxels are red, i.e. near 1.





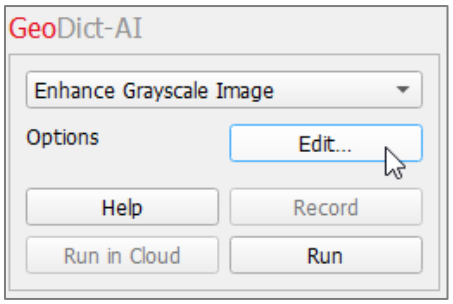
For more visualization options refer to the Visualization handbook of this User Guide.

Enhance Grayscale Image

After producing training data as described on pages 25 and 16 and training a neural network as described starting on page 29, the network can be applied to all images with statistical parameters fitting to the corresponding training data.

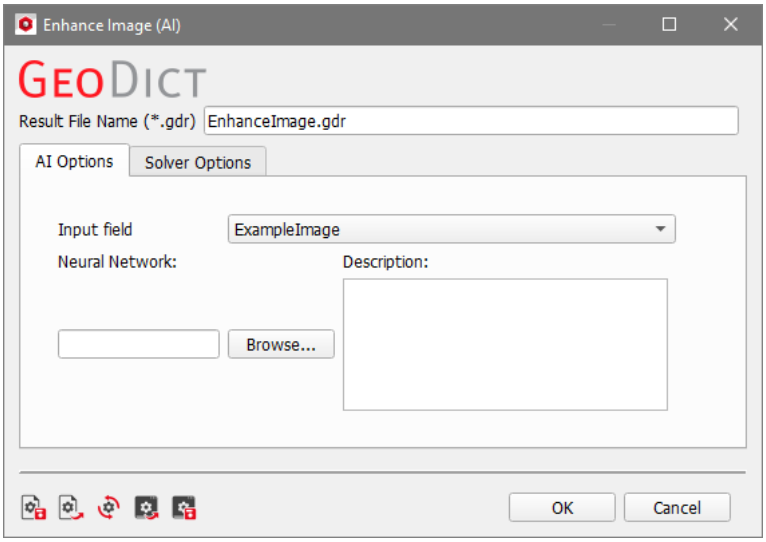
Use GeoDict-AI **Enhance Grayscale Image**, to apply neural networks trained to enhance 3D-scans.

To start, load the image to analyze by selecting **File** → **Load Volume File ...** and browsing for the *.grw image to improve. Then, select **Enhance Grayscale Image** from the pull-down menu in the GeoDict-AI section. If the image is loaded with ImportGeo-Vol, use **Enhance Image (AI)** in the **Image Processing** dialog instead, as described in the ImportGeo-Vol handbook of this User Guide. The image filter has the same functionality and the same parameters.



Click the **Options**' **Edit...** button.

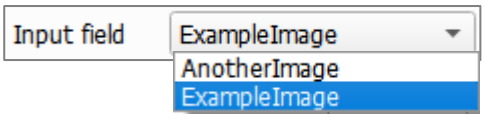
The **Enhance Image (AI)** dialog is organized in two tabs: **AI Options** and **Solver Options**.



AI-Options

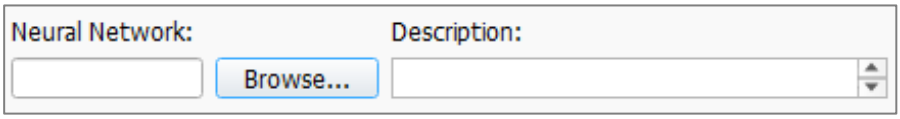
Input Field

Select the **Input field** to enhance from the pull-down menu.

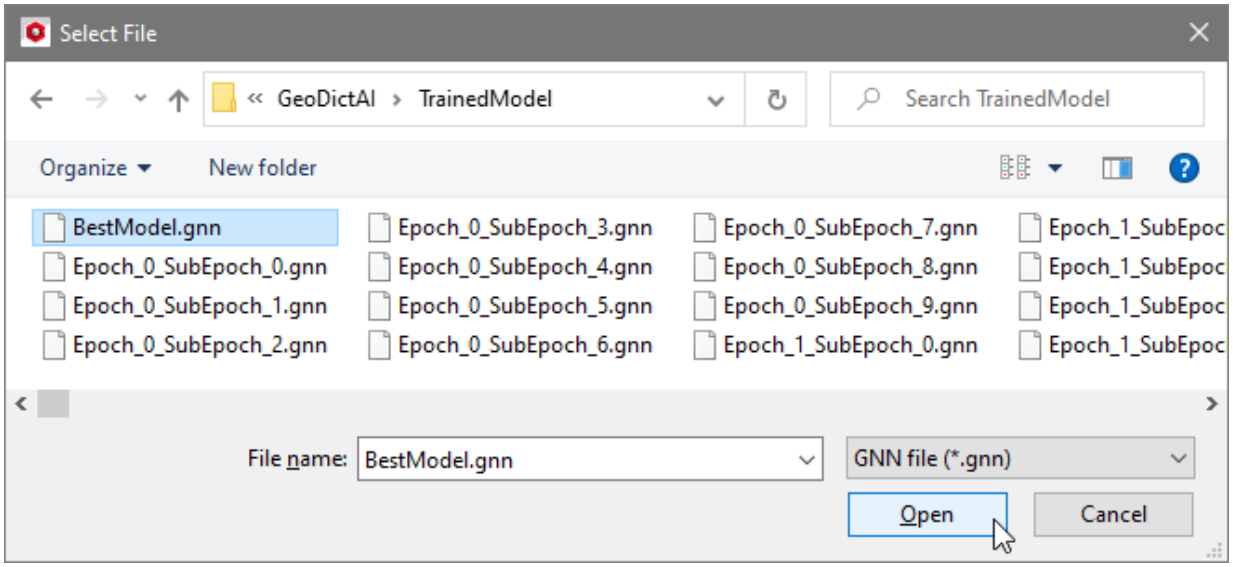


Neural Network and Description

Browse for the neural network trained with **Train Neural Network** and best suitable for the image under consideration. Therefore, under **Neural Network** click **Browse**.



Select **BestModel.gnn** or **FinalModel.gnn**. The difference between these two models is described on page 41, but usually using **BestModel.gnn** is recommended.



In the **Description**, the current constraints for the application of the neural network are listed, as entered in the **Train Neural Network** dialog described on page 30, e.g. the kind of images the neural network was trained for.

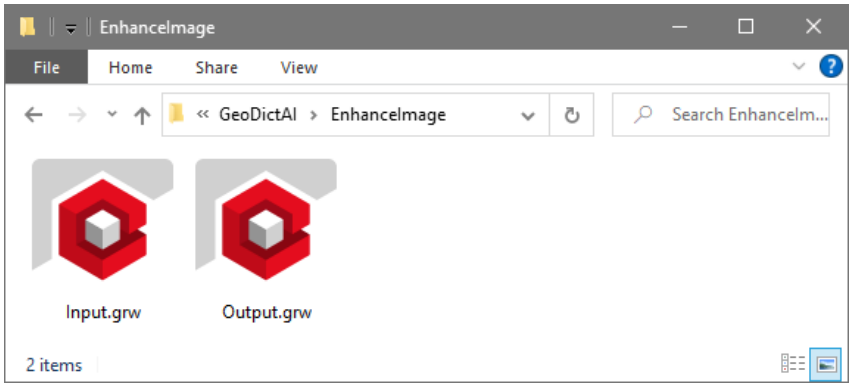
Solver Options

If a suitable graphics card is detected during installation of GeoDict, the GPU mode is used for running GeoDict-AI, otherwise it is running in CPU mode. The options in this tab are the same as for **Apply Neural Network** described starting on page 49.

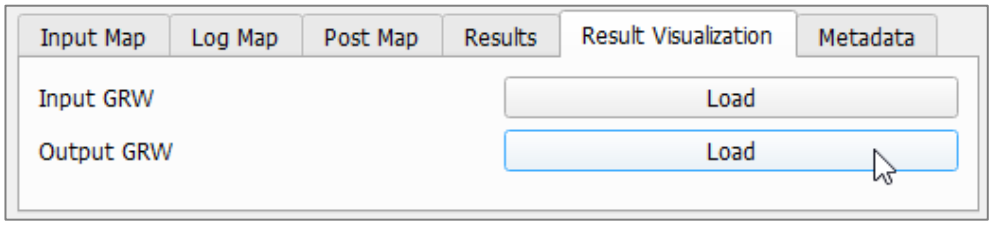
RESULTS

After applying the neural network, a folder with the same name as the GeoDict result file is saved in the chosen project folder (**File → Choose Project Folder...** in the menu bar).

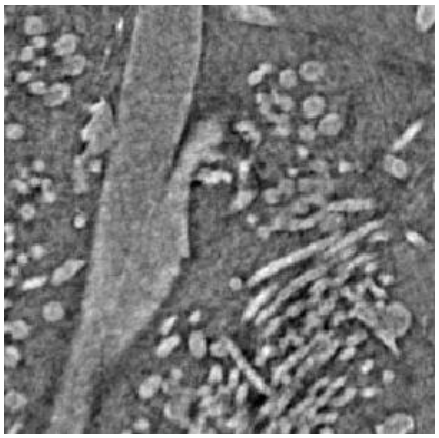
This folder contains the original image (**Input.grw**) and the resulting enhanced image (**Output.grw**).



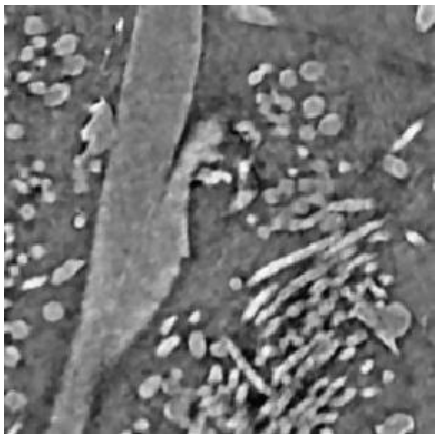
The GeoDict **Result Viewer** opens for the result file and while the report only provides information on runtime and memory usage from the **Result Visualization** tab the input and output files can be loaded by clicking **Load**.



Compare the input and output images to validate the performance of the selected network.



Input GRW

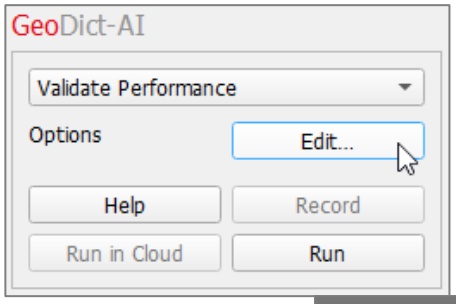


Output GRW

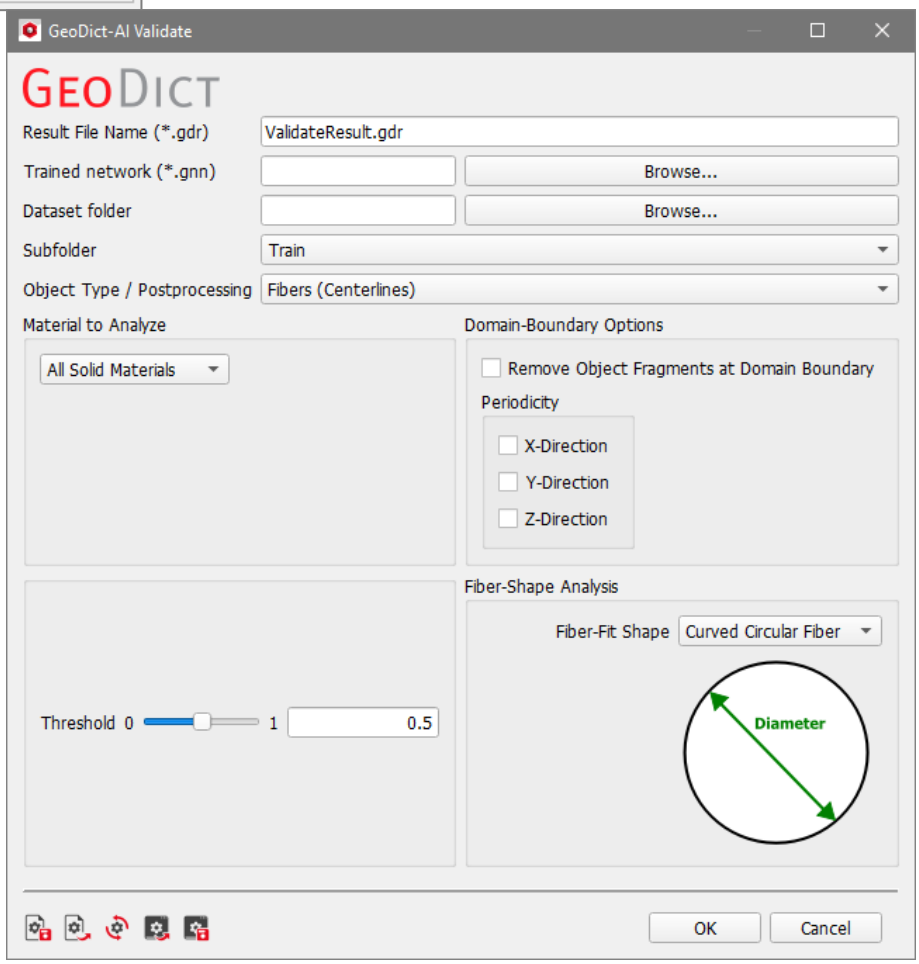
For visualization options refer to the Visualization handbook.

VALIDATE PERFORMANCE

After training a neural network **Validate** it's **Performance** before applying it to new structures. In the GeoDict-AI section select **Validate Performance** from the pull-down menu. The **GeoDict-AI Train Neural Network Options** dialog opens when clicking the **Options' Edit...** button in the GeoDict-AI section.



At the top of the **GeoDict-AI Validate** dialog, the name for the files containing the training results can be entered in the **Result File Name (*.gdr)** box. The default name can be kept, or a new name can be chosen fitting the current project.



Trained Network

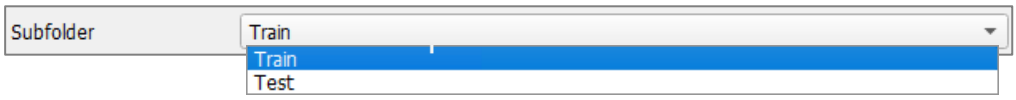
Browse for the **Trained Network (*.gnn)** to validate. Select the **BestModel.gnn** or the **FinalModel.gnn** generated with **Train Neural Network** as described starting on page 29.

Dataset Folder and Subfolder

Choose the folder created with **Create Training Data** containing the corresponding training data in the two folders **Train** and **Test**.

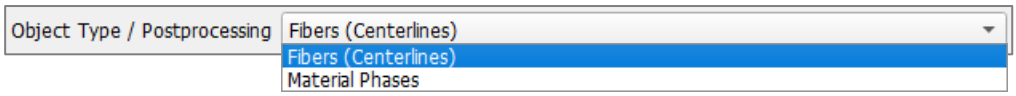
Then select the **Subfolder** that should be used for validation:

- **Train** contains the data that was used for training. The neural network should perform well on this data, as it was explicitly trained for this.
- **Test** contains the data produced explicitly for the validation. This was not used in training and contains different data with similar statistical properties. If the network also performs well on this data, it can be applied on real scans with the required properties.



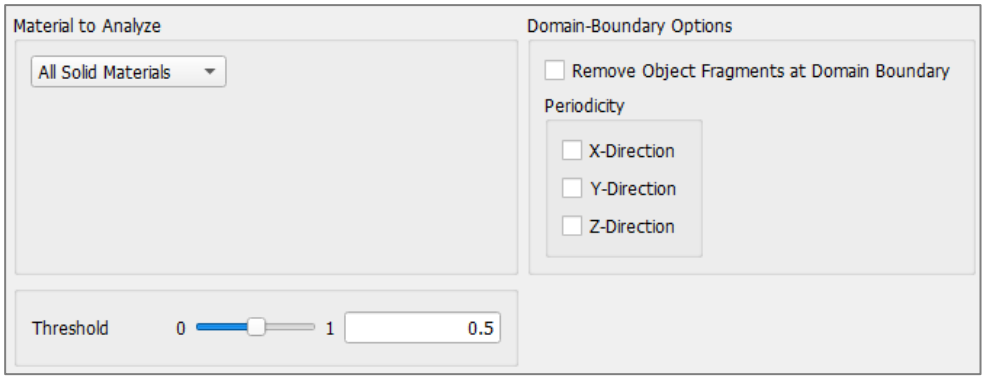
Object Type / Postprocessing

Define the **Object Type / Postprocessing** that was used for creating the training data. Select **Fibers (Centerlines)** if the network was trained to identify the individual fibers. If instead different materials as for example fibers and binder are distinguished, select **Material Phases**. Fibers (Centerlines) can only be applied, if a valid FiberFind license is available.



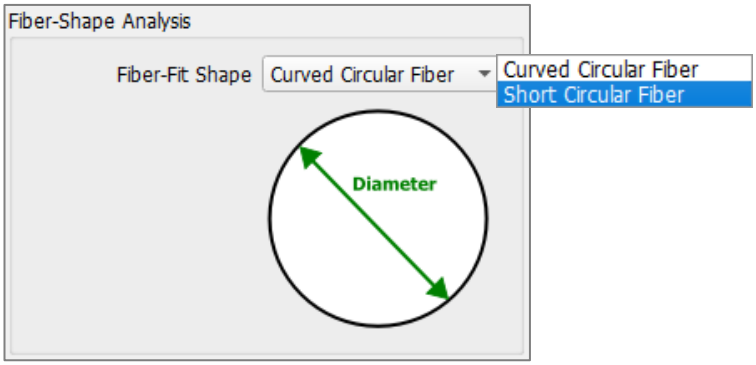
Material to Analyze, Domain-Boundary Options and Threshold

The parameters **Material to Analyze**, **Domain-Boundary Options** and **Threshold** define how the neural network will be applied and are already described starting on page 46.



Fiber-Fit Shape

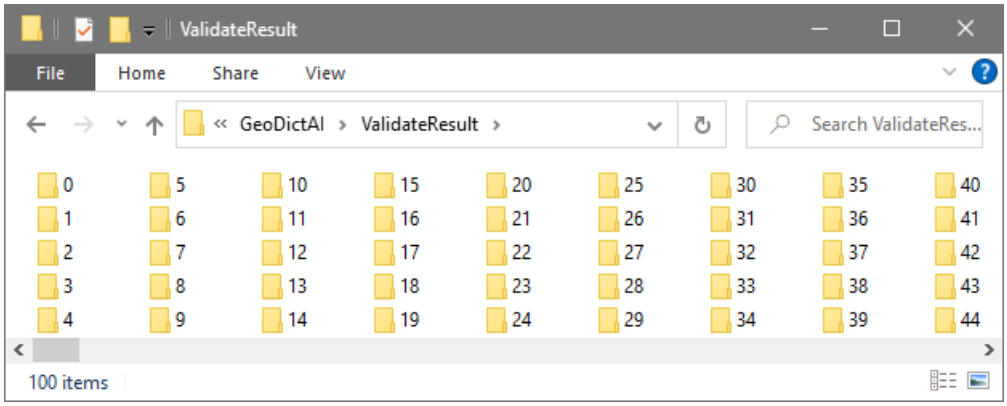
If **Fibers (Centerlines)** is selected in the bottom right corner select the **Fiber-Fit Shape** from the pull-down menu, depending on the training data. **Curved Circular Fiber** and **Short Circular Fiber** are available.



RESULTS

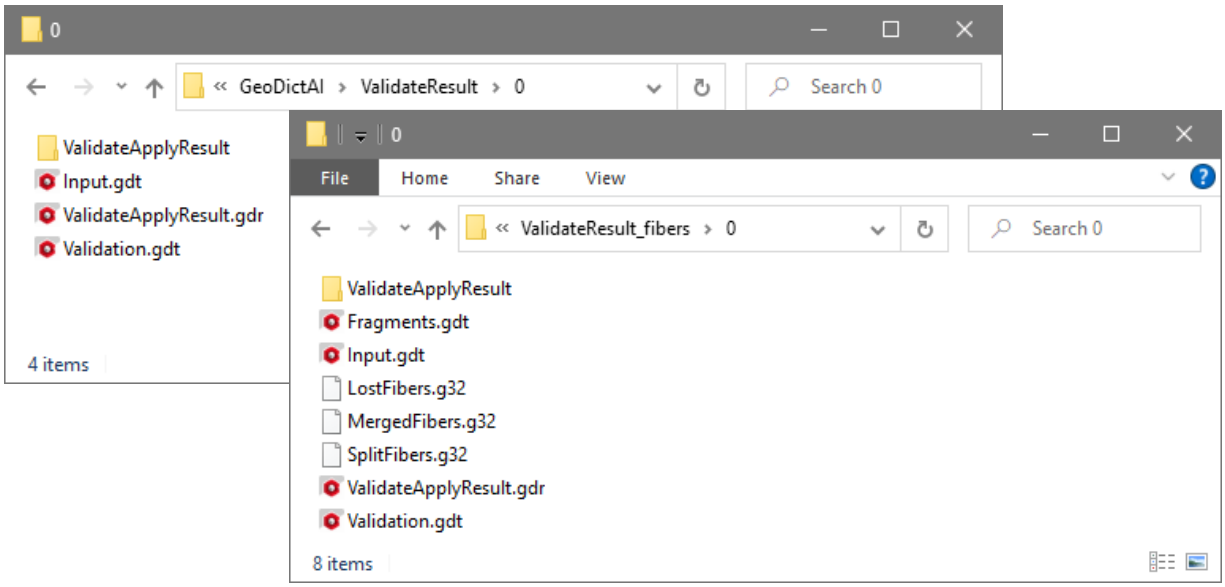
Running **Validate Performance** produces a folder with the same name as the GeoDict result file. Both are saved in the chosen project folder (**File** → **Choose Project Folder...** in the menu bar).

The result folder contains the same number of folders as the folder selected for **Subfolder** as described above, each folder corresponding to a different training data set.

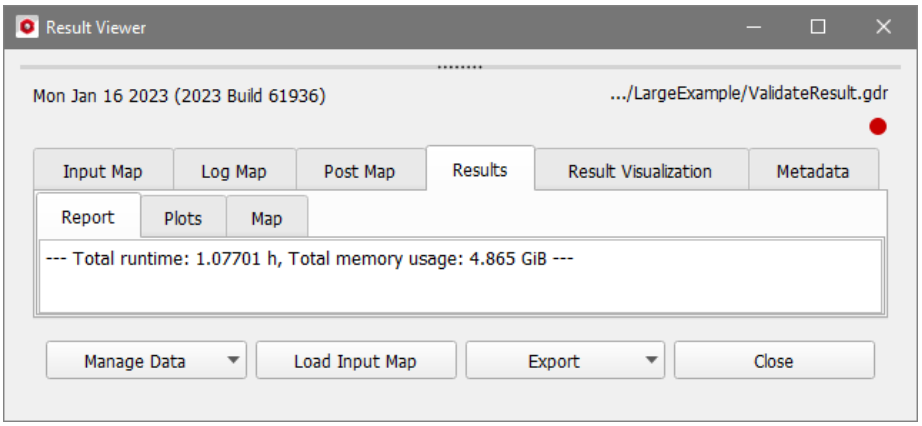


These folders then contain the following data:

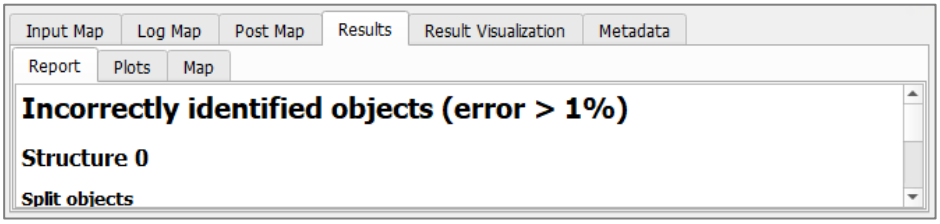
- The file **Input.gdt** is copied from the training data (see page 26).
- The file **ValidateApplyResult.gdr** and the corresponding folder **ValidateApplyResult** are generated by applying the neural network to Input.gdt.
- The file **Validation.gdt** contains the validation result visualization as described on page 64.
- The file **Fragments.gdt** is only available for the **Object Type** option **Fibers**. It contains the fragments not assigned to a fiber.
- The file **LostFibers.g32** is only available for the **Object Type** option **Fibers**. It contains the fibers not detected.
- The file **MergedFibers.g32** is only available for the **Object Type** option **Fibers**. It contains the wrongly merged fibers.
- The file **SplitFibers.g32** is only available for the **Object Type** option **Fibers**. It contains the wrongly split fibers.



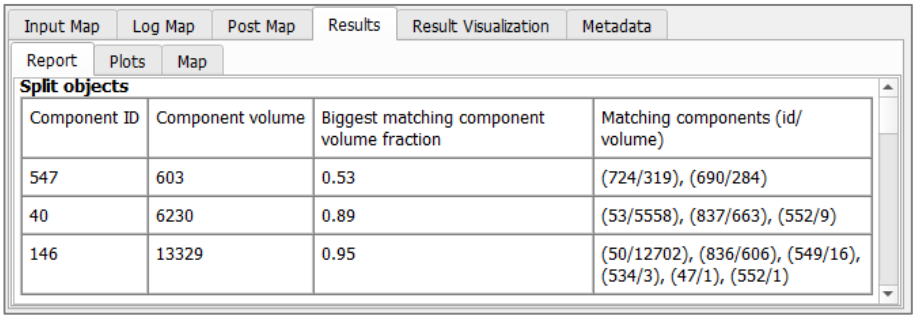
The GeoDict **Result Viewer** opens for the result file and the **Results – Report** subtab only shows the default report if **Material Phases** was chosen for **Object Type**.



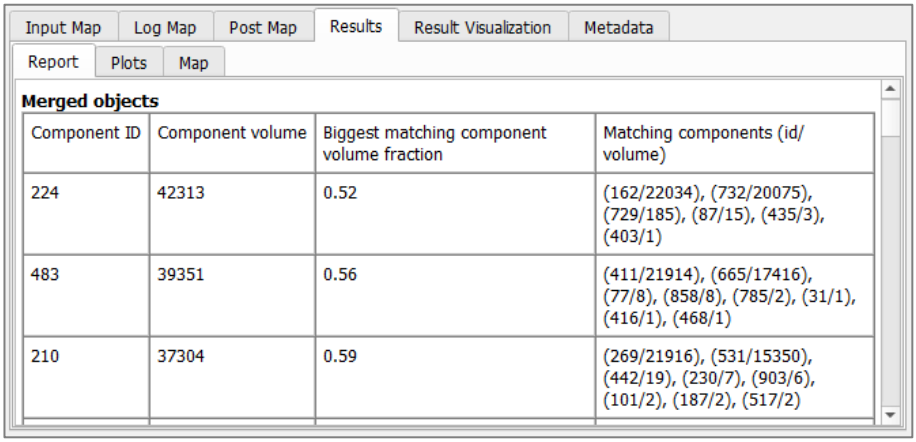
If **Fibers** was chosen for **Object Type**, the report additionally shows statistics regarding the incorrectly identified fibers for each structure used in the validation.



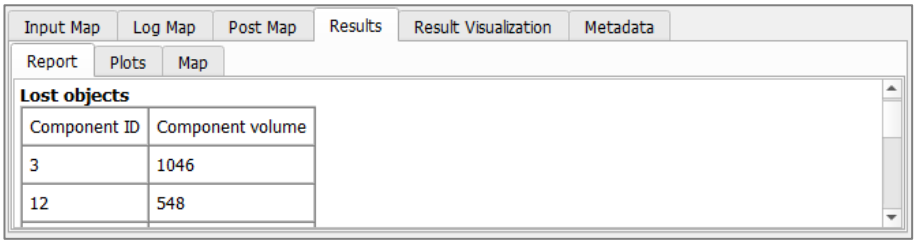
Find information about the **Split objects**, fibers that were incorrectly split in two or more fibers.



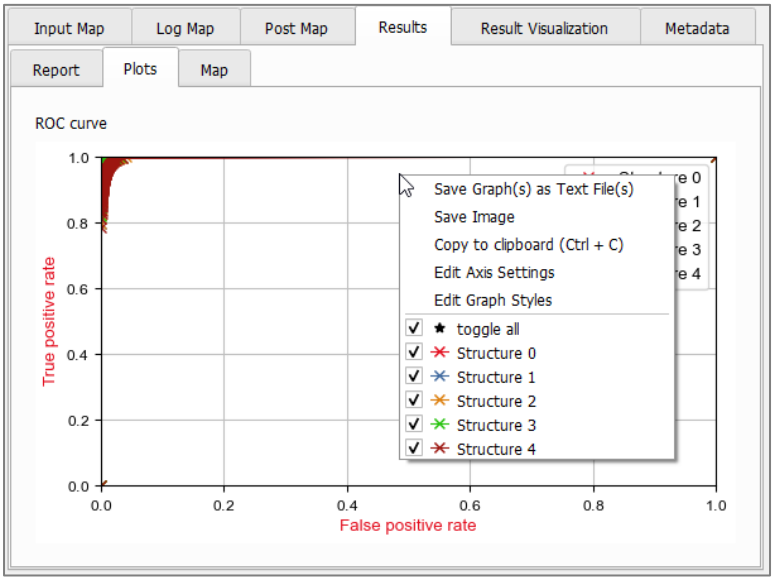
Next find a table for the **Merged objects**, fibers incorrectly merged into one.



Also the **Lost Objects** are listed. These are fibers, where parts of them were not detected.



The **Plot** subtab, however, shows the **Receiver Operating Characteristic** (**ROC**) curve for each selected structure. Right-clicking in the plot opens a dialog offering many possibilities to change the plot settings or to save the data. Refer to the Result Viewer handbook of this User Guide to learn more about these options.



When applying a neural network, a confidence field is generated. This field then is segmented with the **Threshold** given in the **Apply Neural Network Options** dialog, described on page 47.

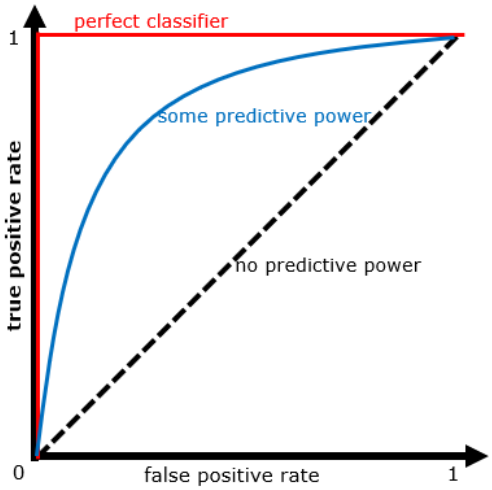
Usually, not all voxels are identified correctly. The proportion of voxels correctly assigned to the target material, then is called the **True positive rate**. The proportion of voxels wrongly assigned to the target material is called the **False positive rate**.

For a threshold of 0 all solid voxels are assigned to the target material. Thus, the **True positive rate** and the **False positive rate** are both 1. For a threshold of 1 no voxels are assigned to the target material and thus, both rates are 0.

The **ROC curve** shows how the relation of **True positive rate** and **False positive rate** changes if varying the threshold from 0.0 to 1.0.

If the curves look as above, the neural network performs very well on the structures.

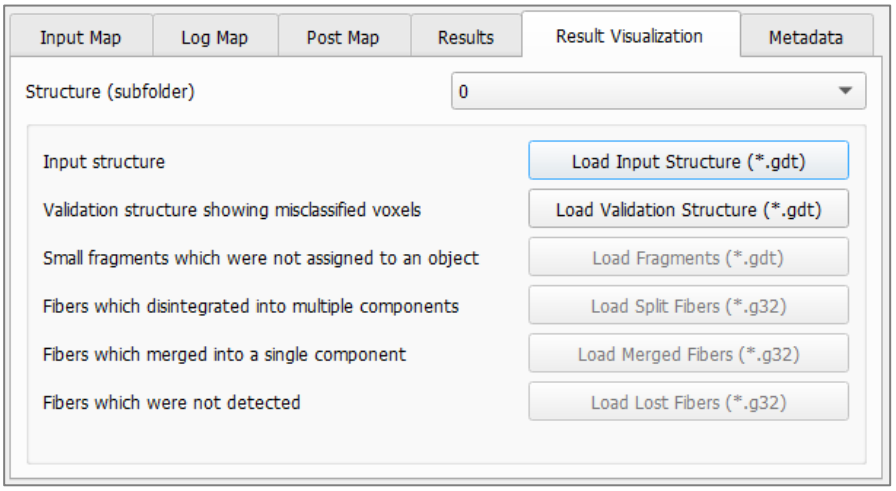
The optimum threshold is the upper left area, as the true positive rate is high, but the false positive rate is still low.



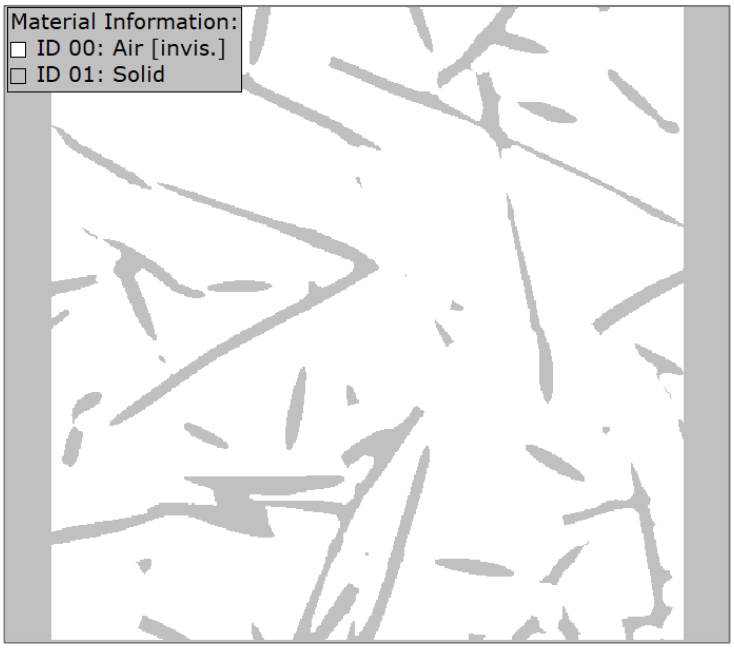
The values should lie above of the main diagonal, as otherwise the performance is not better than simply guessing the material IDs. If for all thresholds in (0,1) the true positive rate is 1 and the false positive rate is 0, the network is a perfect classifier, i.e. the result is always perfectly identified materials for all possible thresholds. This means, that the confidence field only contains values near 0 and values near 1, as the neural network is very confident. Usually, obtaining a perfect classifier is not realistic. But it can be close, as in the example above.

The **Result Visualization** tab provides several 3D visualization options for the results. From the pull-down menu **Structure (subfolder)** select the subfolder from which the results should be visualized.

If **Material Phases** was selected for **Object Type / Postprocessing** in the panel below only the **Input structure** and the **Validation structure showing misclassified voxels** are available.



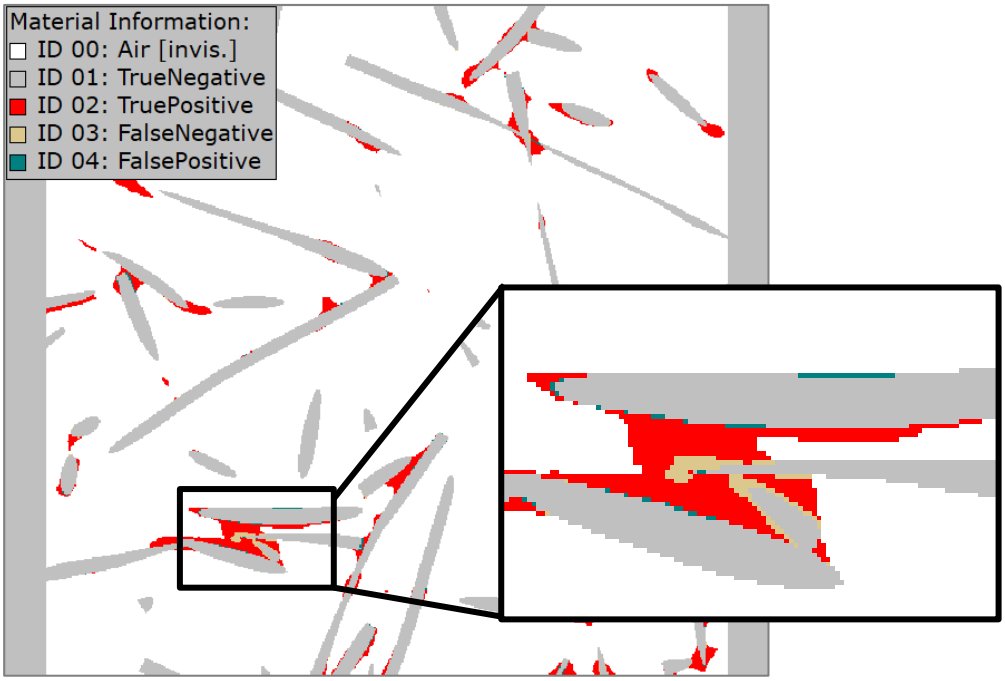
Clicking **Load Input Structure (*.gdt)** loads the file **Input.gdt** only containing two material IDs: ID 00 for the background voxels, usually the pore space, and ID 01 for the solid materials.



Click **Load Validation Structure (*.gdt)** to load the file **Validation.gdt**. Five material IDs are present in this structure:

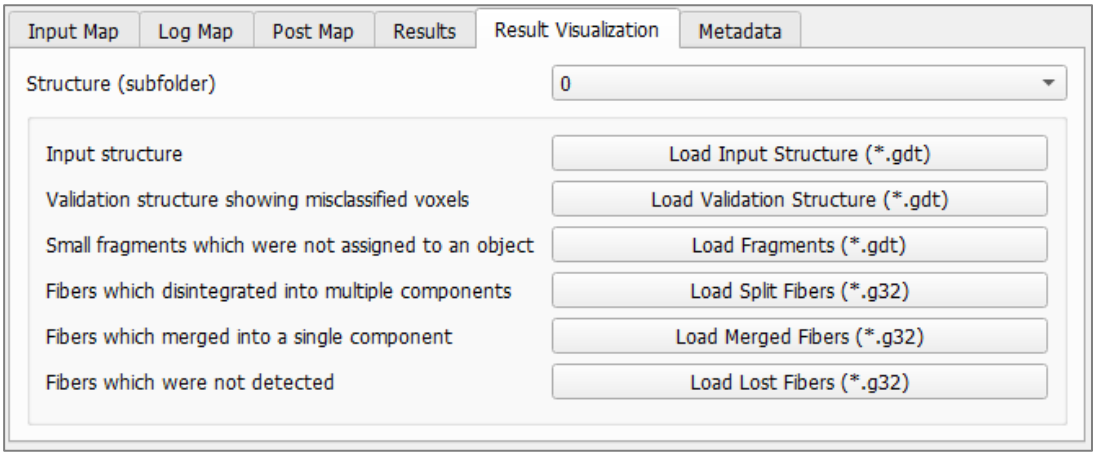
- **ID 00 Pore**: The pore space that is not taken into account for the identification of target material, in the example it is assigned to air, according to the material IDs in the input structure.
- **ID 01 TrueNegative**: the solid voxels that the neural network correctly identified as no target material.
- **ID 02 TruePositive**: the solid voxels that are correctly identified as target material.
- **ID 03 FalseNegative**: the solid voxels that are target material, but are not identified as it.
- **ID 04 FalsePositive**: the solid voxels that are wrongly identified as target material.

In the image below, the material phases binder and fibers were distinguished. The voxels correctly identified as fiber (**TrueNegative**) are visualized in gray and the correct identified binder in red (**TruePositive**). The beige voxels show, where binder was identified as fiber (**FalseNegative**) and the fibrous voxels identified as binder are marked in blue (**FalsePositive**). In this example, the neural network identified the binder really well. Only a few voxels at the boundary between fiber and binder were identified wrongly.

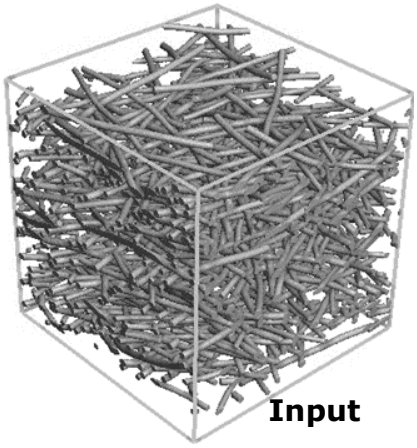


If **Fibers** was selected for **Object Type**, additionally files are available:

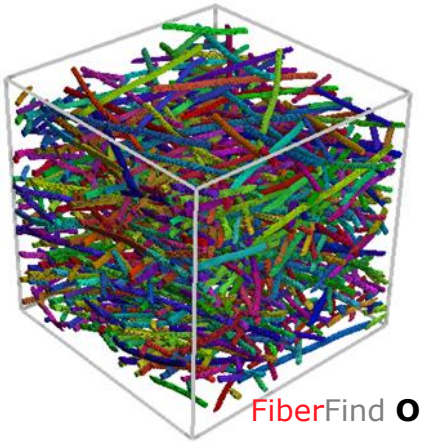
- **Small fragments which were not assigned to an object**
- **Fibers which disintegrated into multiple components**
- **Fibers which merged into a single component**.
- **Fibers which were not detected**



In the following figure, the input fiber structure and the result from FiberFind are shown. The FiberFind result files and result folders can also be found in the **Validate Performance** result folder.

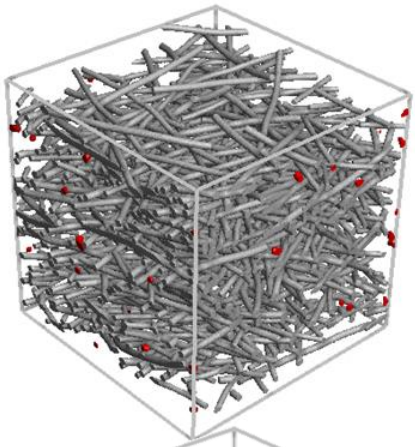


Input

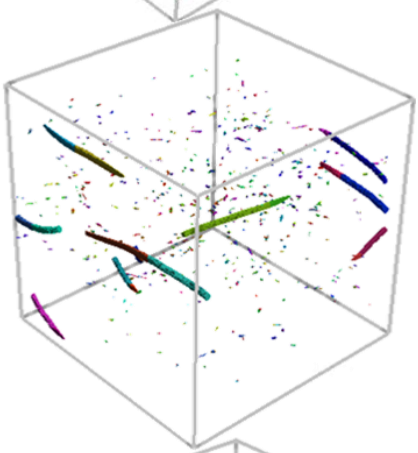


FiberFind **Output**

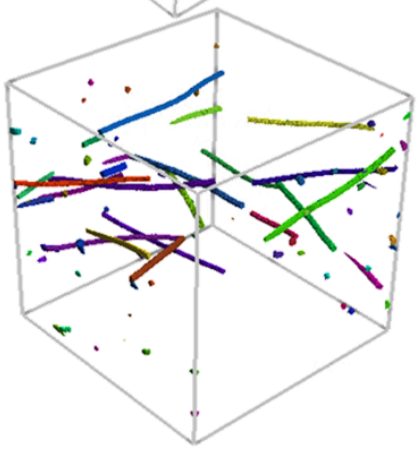
To check, if there are fragments of solid material not identified as fibers, click **Load Fragments (*.gdt)**.



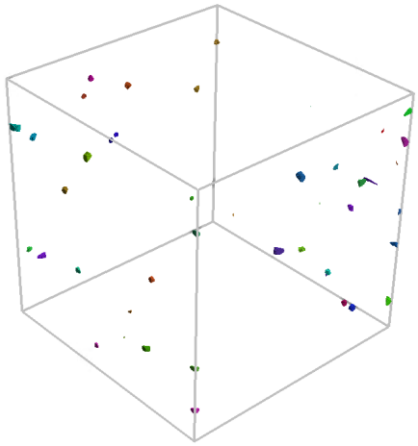
Click **Load Split Fibers (*.g32)** to observe if fibers were split into multiple components.



Also, different fibers can have been merged into one fiber. Click **Load Merged Fibers (*.g32)** to check on it.



There can be fiber parts not detected as fibers. Click **Load Lost Fibers (*.g32)** to visualize them.



REFERENCES

[1] Ronneberger, O., Fischer, P., Brox, T., U-Net: Convolutional Networks for Biomedical Image Segmentation, arXiv:1505.04597 (2015)

[2] Kingma, Diederik P., and Jimmy Ba. Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014)

[3] Ruder, S., An overview of gradient descent optimization algorithms, arXiv:1609.04747 (2016)

[4] Sobol, I.M.: On the Distribution of points in a cube and the approximate evaluation of integrals, USSR Computational Mathematics and Mathematical Physics 7: 86-112 (1967)

https://doi.org/10.30423/userguide.geodict

Technical documentation:

Janine Hilden
Rolf Westerteiger
Olga Lykhachova
Barbara Planas

MATH 2 MARKET

Math2Market GmbH
Richard-Wagner-Str. 1, 67655 Kaiserslautern, Germany
www.geodict.com