Simple TF2 + Keras model for segmentation (to detect cell nuclei)

This notebook the entire workflow of training an ANN with TensorFlow 2 (https://www.tensorflow.org/) using the keras API and exporting the trained model to the CZModel format (https://github.com/zeiss-microscopy/OAD/blob/master/Machine_Learning/docs/ann_model_specification.md) to be ready for use within the Intellesis (https://www.zeiss.de/mikroskopie/produkte/mikroskopsoftware/zen-intellesis-image-segmentation-by-deep-learning.html) infrastructure.

- The trained model is rather simple (for demo purposed) and trained on a small test dataset.
- **Therefore, this notebook is meant to be understood as a guide for exporting trained models**
- **The notebook does not show how train a model correctly.**

In [2]:

```
# required imports to train a simple TF2 + Keras model for segmentation and package it a
s CZMODEL
# the CZMODEL can be then imported in ZEN and used for segmentation and image analysis w
orkflows

# general imports
import os
import tensorflow as tf
import numpy as np

# those functions are provided by the PyPi package called czmodel (by ZEISS)
from czmodel.util.preprocessing import PerImageStandardization
from czmodel.model_metadata import ModelMetadata, ModelSpec
from czmodel import convert_from_model_spec, convert_from_json_spec
```

Training Pipeline

This section describes a simple training procedure that creates a trained Keras model.

- Therefore, it only represents the custom training procedure
- Such procedure will vary from case to case and will contain more sophisticated ways to generate an optimized Keras model

In [4]:

```
# Define the parameters for loading the training data

# place the original *.png images here
IMAGES_FOLDER = 'data/nuclei_images/'

# place the respective label *.png images here
# masks images have one channel (0=background and 1=nucleus)
MASKS_FOLDER = 'data/nuclei_masks/'

# define the number of channels
# this means using a grayscale image with one channel only
CHANNELS = 1
```

In [5]:

```
# Read the images
# This part contains the logic to read pairs of images and label masks for training !

# the the sample images
sample_images = sorted([os.path.join(IMAGES_FOLDER, f) for f in os.listdir(IMAGES_FOLDER
)
                        if os.path.isfile(os.path.join(IMAGES_FOLDER, f))])

# get the maks
sample_masks = sorted([os.path.join(MASKS_FOLDER, f) for f in os.listdir(MASKS_FOLDER)
                       if os.path.isfile(os.path.join(MASKS_FOLDER, f))])

# load images as numpy arrays
images_loaded = np.asarray([tf.image.decode_image(tf.io.read_file(sample_path), channels
=CHANNELS).numpy()
                            for sample_path in sample_images])

# load labels as numpy arrays
masks_loaded = np.asarray([tf.one_hot(tf.image.decode_image(tf.io.read_file(sample_path
), channels=1)[...,0], depth=2).numpy()
                           for sample_path in sample_masks])
```

Define a simple model

This part defines a simple Keras model with two convolutional layers and softmax activation at the output node. It is also possible to add pre.processing layers to the model here.

In order to make the model robust to input scaling we standardize each image before training with the PerImageStandardization layer provided by the `czmodel` package.

In [6]:
```
# Define simple Keras model with two convolutional layers and softmax activation at the
 output node

model = tf.keras.models.Sequential([PerImageStandardization(input_shape=(128, 128, 1)),
                                    tf.keras.layers.Conv2D(16, 3, padding='same'),
                                    tf.keras.layers.Conv2D(2, 1, activation='softmax', p
adding='same')])

# compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['categorical_a
ccuracy'])
```

Fit the model to the loaded data

This part fits the model to the loaded data and evaluates it on the training data. In this test example we do not care about an actual evaluation of the model using validation and test datasets.

In [7]:

```
# define number of training epochs
num_epochs = 10

# fit the model to the data
model.fit(images_loaded, masks_loaded,
          batch_size=32,
          epochs=num_epochs)

# get the loss and acuary values
loss, accuracy = model.evaluate(images_loaded, masks_loaded)

# show the final accuracy achieved
print("The model achieves {}% accuracy on the training data.".format(accuracy * 100))
```

```
Train on 200 samples
Epoch 1/10
200/200 [==============================] - 1s 6ms/sample - loss: 0.6216 - categorical
_accuracy: 0.8455
Epoch 2/10
200/200 [==============================] - 1s 4ms/sample - loss: 0.5985 - categorical
_accuracy: 0.8503
Epoch 3/10
200/200 [==============================] - 1s 4ms/sample - loss: 0.5765 - categorical
_accuracy: 0.8548
Epoch 4/10
200/200 [==============================] - 1s 4ms/sample - loss: 0.5532 - categorical
_accuracy: 0.8597
Epoch 5/10
200/200 [==============================] - 1s 4ms/sample - loss: 0.5326 - categorical
_accuracy: 0.8666
Epoch 6/10
200/200 [==============================] - 1s 4ms/sample - loss: 0.5095 - categorical
_accuracy: 0.8697
Epoch 7/10
200/200 [==============================] - 1s 4ms/sample - loss: 0.4869 - categorical
_accuracy: 0.8715
Epoch 8/10
```

```
Epoch 8/10
200/200 [==============================] - 1s 4ms/sample - loss: 0.4643 - categorical
_accuracy: 0.8755
Epoch 9/10
200/200 [==============================] - 1s 4ms/sample - loss: 0.4434 - categorical
_accuracy: 0.8805
Epoch 10/10
200/200 [==============================] - 1s 4ms/sample - loss: 0.4211 - categorical
_accuracy: 0.8829
200/200 [==============================] - 0s 2ms/sample - loss: 0.4081 - categorical
_accuracy: 0.8820
The model achieves 88.20095658302307% accuracy on the training data.
```

Create a CZModel from the trained Keras model

In this section we export the trained model to the CZModel format using the czmodel library and some additional meta data all possible parameter choices are described in the ANN model specification (https://github.com/zeiss-microscopy/OAD/blob/master/Machine_Learning/docs/ann_model_specification.md).

Define Meta-Data

We first define the meta-data needed to run the model within the Intellesis infrastructure. The `czmodel` package offers a named tuple `ModelMetadata` that allows to either parse as JSON file as described in the specification document (https://github.com/zeiss-microscopy/OAD/blob/master/Machine_Learning/docs/ann_model_specification.md) or to directly specify the parameters as shown below.

Create a Model Specification Object

The export functions provided by the `czmodel` package expect a `ModelSpec` tuple that features the Keras model to be exported and the corresponding model meda-data.

Therefore, we wrap our model and the `model_metadata` instane into a `ModelSpec` object.

In [8]:

```
# define the model metadata
model_metadata = ModelMetadata.from_params(name='Simple_Nuclei_SegmentationModel',
                                           color_handling='ConvertToMonochrome',
                                           pixel_type='Gray16',
                                           classes=["Background", "Nucleus"],
                                           border_size=8)


# Create a model specification object used for conversion
model_spec = ModelSpec(model=model, model_metadata=model_metadata)

# Define dimensions - ZEN Intellesis requires fully defined spatial dimensions.
# This is the tile size used by the ZEN TilingClient to pass an image to the segmentatio
n service.

# Important: The tile size has to sufficiently small to be handled by your hardware!
# Optional: Define target spatial dimensions of the model for inference.
spatial_dims = (1024, 1024)  # Optional: Target spatial dimensions of the model for infe
rence.
```

Perform model export into *.czmodel file format

The `czmodel` library offers two functions to perform the actual export.

- `convert_from_json_spec` allows to provide a JSON file with all information to convert a model in SavedModel format on disk to a `.czmodel` file that can be loaded with ZEN.
- `convert_from_model_spec` expects a `ModelSpec` object, an output path and name and optionally target spatial dimensions for the expected input of the exported model. From this information it creates a `.czmodel` file containing the specified model.

```
convert_from_model_spec(model_spec=model_spec,
                        output_path=folder_to_store_czmodel,
                        output_name=name_of_the_model,
                        spatial_dims=spatial_dims)
```

In [9]:

```
# perform the actual model export directly into a *.czmodel file

convert_from_model_spec(model_spec=model_spec,
                        output_path='./czmodel_output',
                        output_name='simple_nuclei_segmodel',
                        spatial_dims=spatial_dims)

# In the example above there will be a ""./czmodel_output/simple_nuclei_segmodel.czmode
l" file saved on disk.
```

Remarks

The generated .czmodel file can be directly loaded into ZEN Intellesis to perform segmentation tasks with the trained model. If there is already a trained model in SavedModel format present on disk, it can also be converted by providing a meta-data JSON file as described in the specification (https://github.com/zeiss-microscopy/OAD/blob/master/Machine_Learning/docs/ann_model_specification.md).

The following JSON document describes the same meta-data applied in the use case above:

```
{
"Name": "SimpleNucleiModel From JSON",
"BorderSize": 8,
"ColorHandling": "ConvertToMonochrome",
"PixelType": "Gray16",
"Classes": ["Background", "Nuclei"],
"ModelPath": "saved_tf2_model_output",
}
```

This information can be copied to a file e.g. in the current working directory `./model_spec.json` that also contains the trained model in SavedModel format e.g. generated by the following line:

In [10]:

```
# save the trained TF2.SavedModel as a folder structure
# The folder + the JSON file can be also used to import the model in ZEN (still fas a bu
g)

model.save('./saved_tf2_model_output/')
```

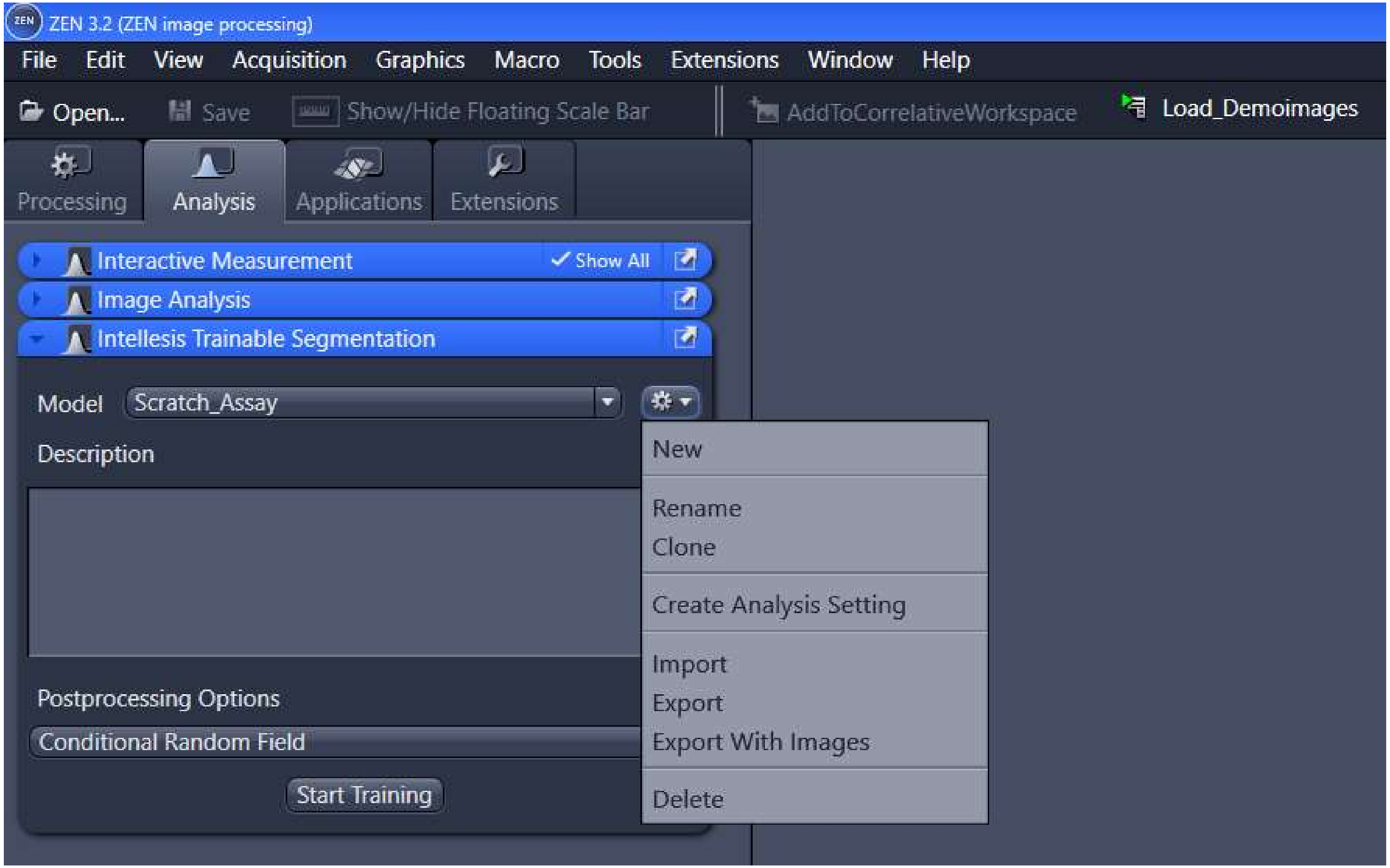
The CZMODEL file (which is essentially a zip file) contains:

- model **guid** file: modelid=e47aabbd-8269-439c-b142-78feec2ed2dd
- model file: modelid=e47aabbd-8269-439c-b142-78feec2ed2dd.model
- model description: e47aabbd-8269-439c-b142-78feec2ed2dd.xml

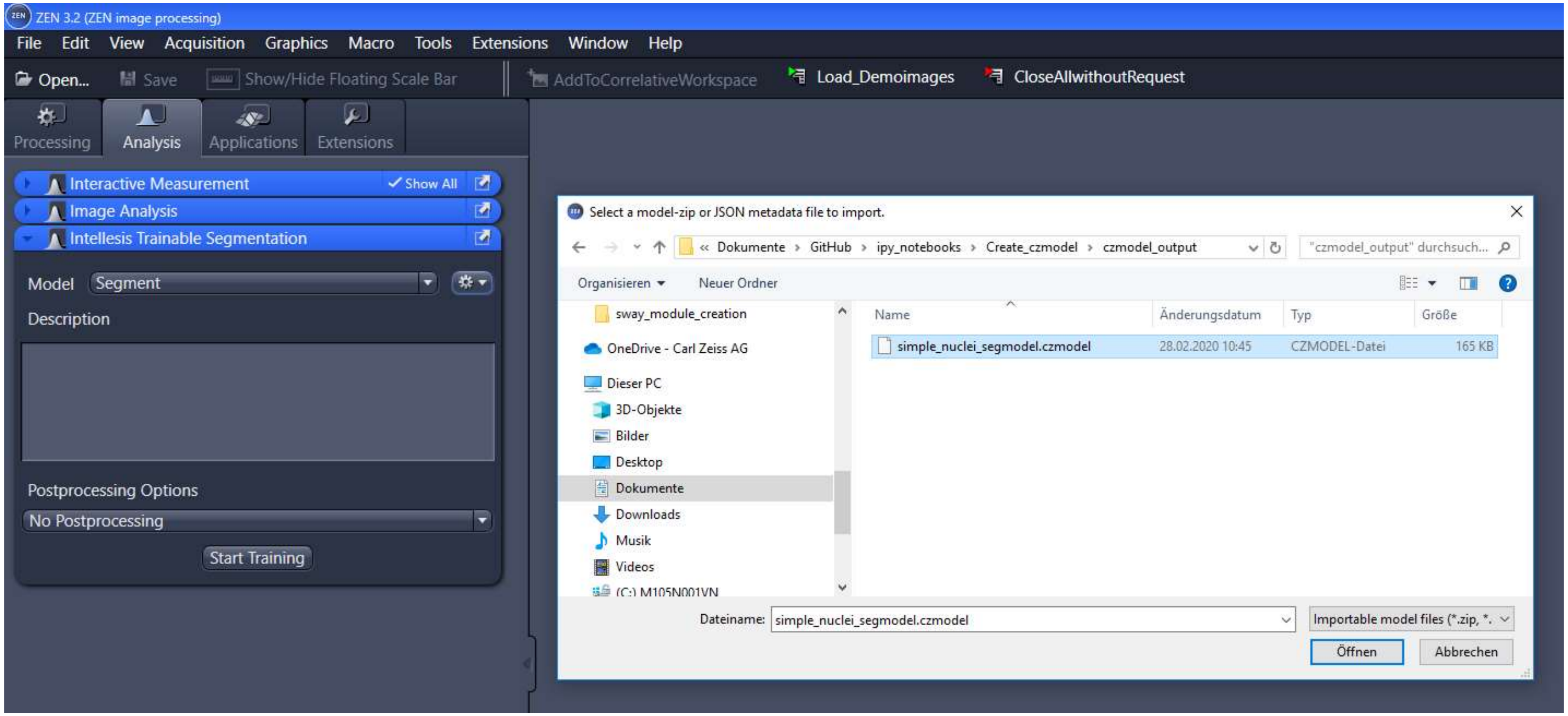
Example of a model XML description

```xml
<?xml version='1.0' encoding='utf-8'?>
<Model Version="3.0.0">
    <Id>e47aabbd-8269-439c-b142-78feec2ed2dd</Id>
    <ModelName>Simple_Nuclei_SegmentationModel</ModelName>
    <Status>Trained</Status>
    <FeatureExtractor>DeepNeuralNetwork</FeatureExtractor>
    <Postprocessing />
    <ColorHandling>ConvertToMonochrome</ColorHandling>
    <Channels>
        <Item PixelType="Gray16" />
    </Channels>
    <TrainingClasses>
        <Item LabelValue="1" Name="Background" colB="0" colG="0" colR="255" />
        <Item LabelValue="2" Name="Nucleus" colB="0" colG="255" colR="0" />
    </TrainingClasses>
    <BorderSize>8</BorderSize>
</Model>
```

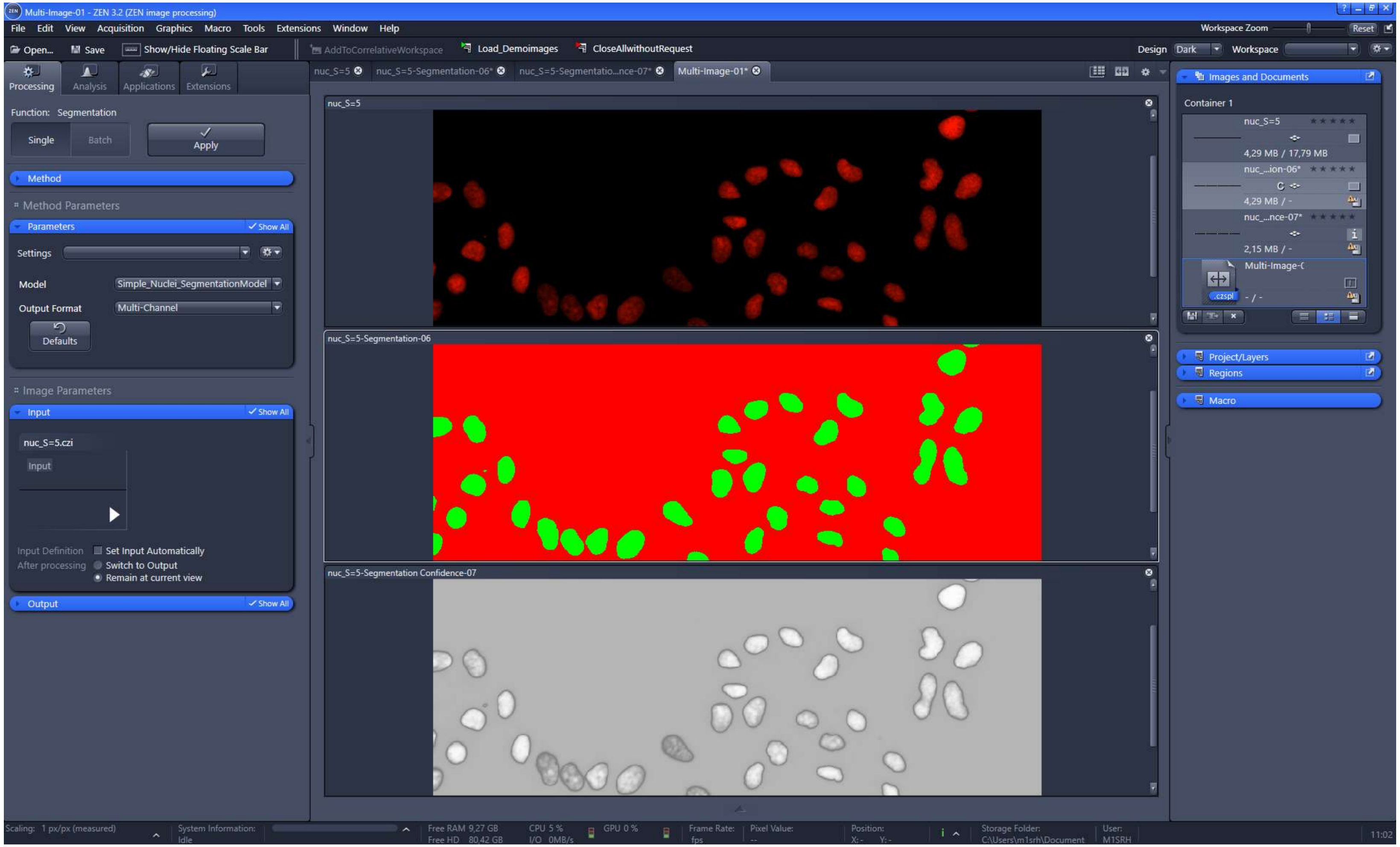
To import the newly created model just use the **`Import`** function of the Intellesis Trainable Segmentation module in ZEN.



Select the **`simple_nuclei_segmodel.czmodel`** file and press the **`Open`** button.



Use the IP-function **`Segmentation`** to segment an image using the imported CZMODEL (containing the trained network).



To use the trained model to analyse an image there are two main options

1. directly create an Image Analysis Setting based on the model (no class hierachy, but very simple)
2. assign the trained model to s specfic class inside a customized image analyssis setting (shown below)

The crucial step (when not using option 1) is the Select the correct **`Class Segmentation Method`** inside the Image Analysis Wizard.

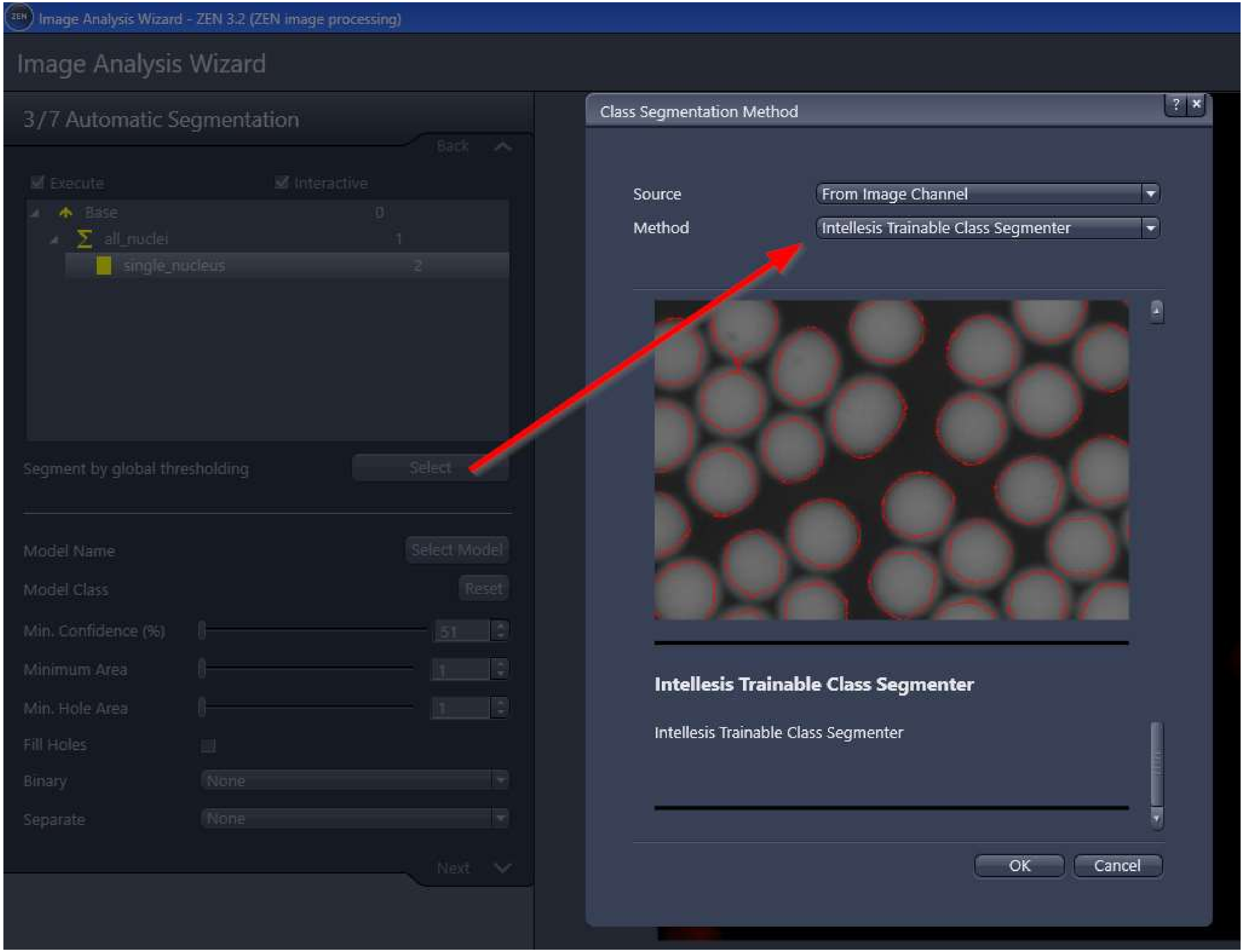


Image Analysis Wizard - ZEN 3.2 (ZEN image processing)
Image Analysis Wizard
3/7 Automatic Segmentation
Back
Execute
Interactive
Base 0
all_nuclei 1
single_nucleus 2
Segment by global thresholding
Select
Model Name
Select Model
Model Class
Reset
Min. Confidence (%)
51
Minimum Area
1
Min. Hole Area
1
Fill Holes
Binary
None
Separate
None
Next
Class Segmentation Method
Source
From Image Channel
Method
Intellesis Trainable Class Segmenter
Intellesis Trainable Class Segmenter
Intellesis Trainable Class Segmenter
OK
Cancel

Use the `Select Model` function to assign the trained model and the actual **class** (from the trained model) of interes to assign the model / class to the respective object inside the image analysis setting.

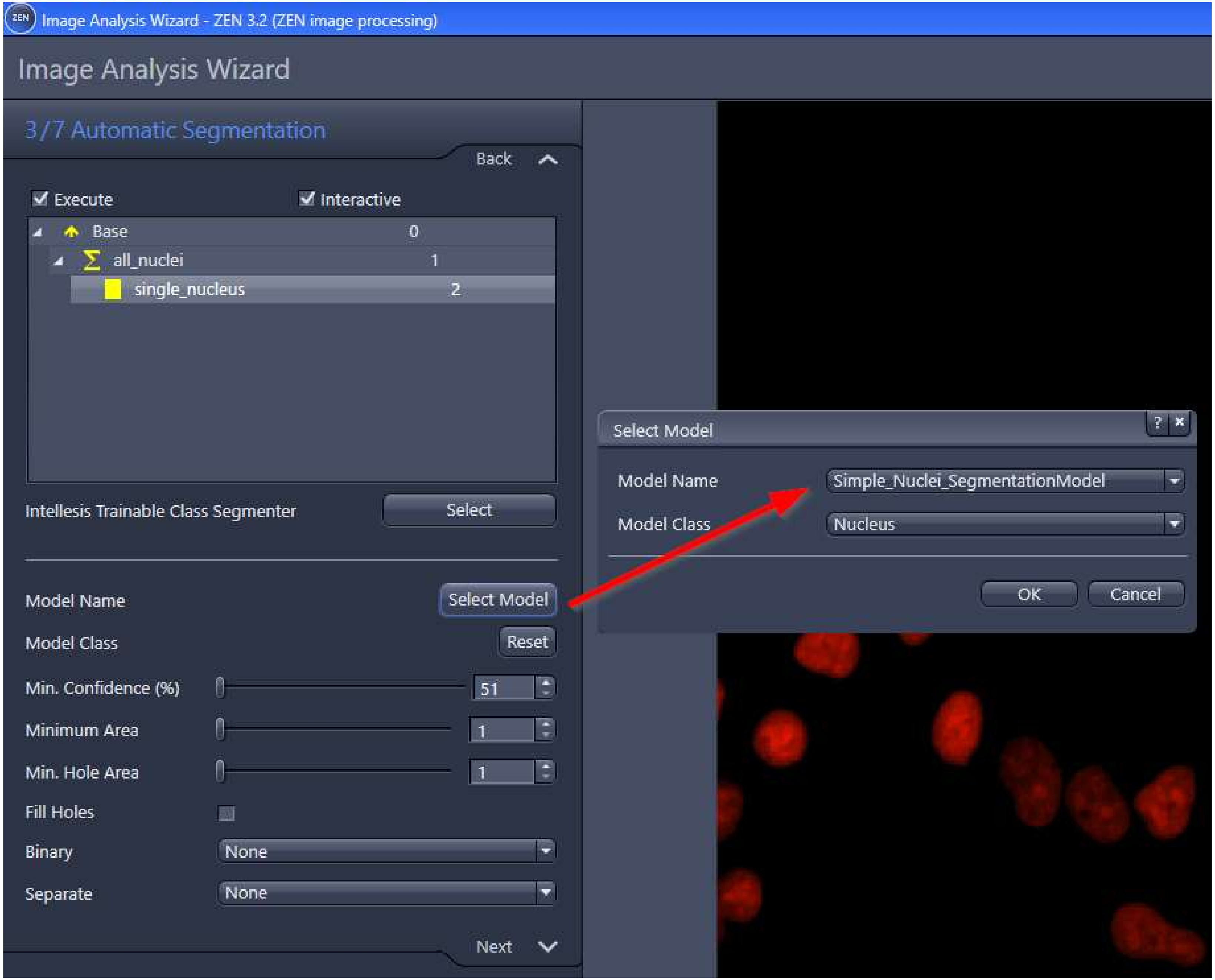
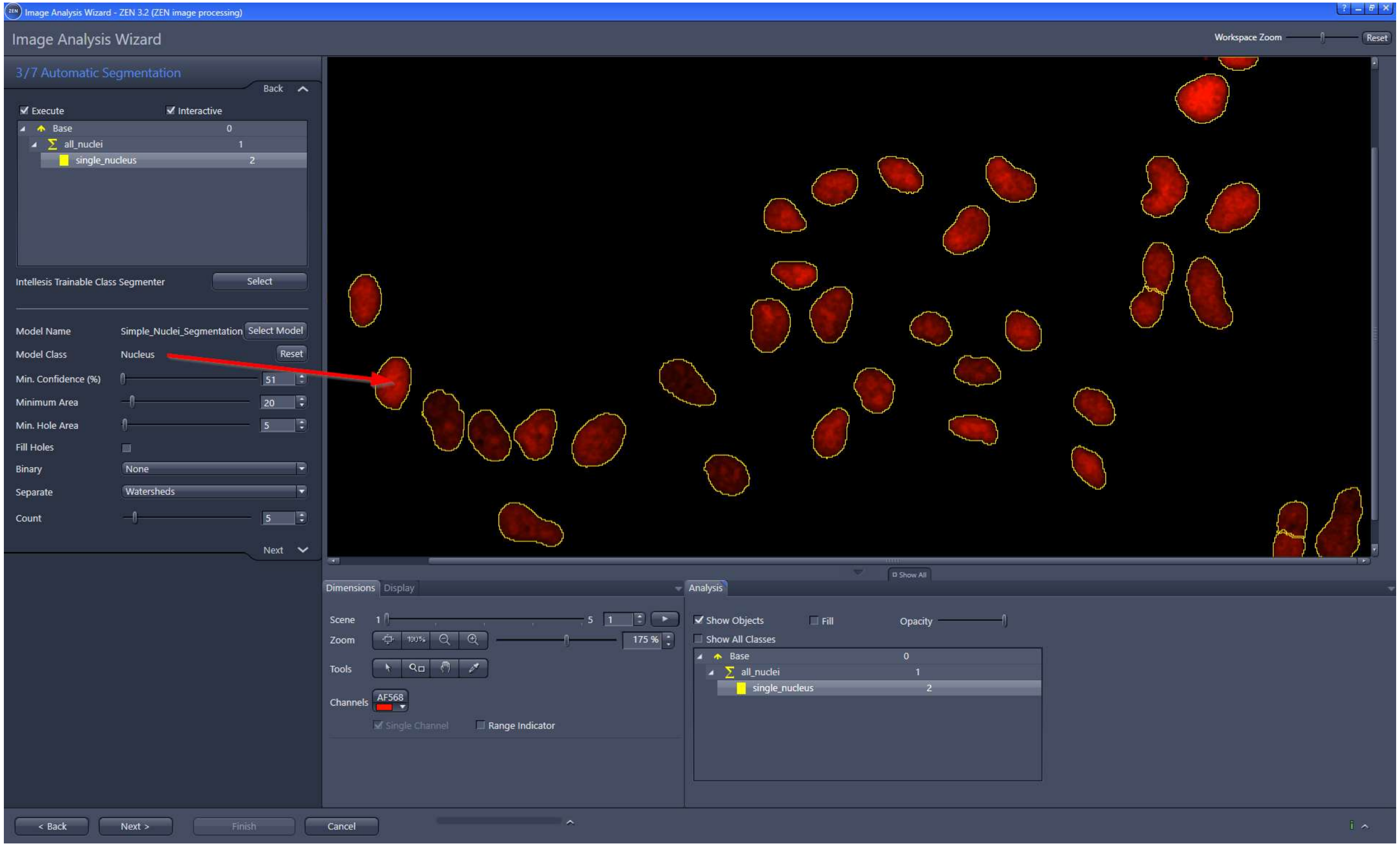


Image Analysis Wizard - ZEN 3.2 (ZEN image processing)
Image Analysis Wizard
3/7 Automatic Segmentation
Back
Execute
Interactive
Base 0
all_nuclei 1
single_nucleus 2
Intellesis Trainable Class Segmenter
Select
Model Name
Select Model
Model Class
Reset
Min. Confidence (%)
51
Minimum Area
1
Min. Hole Area
1
Fill Holes
Binary
None
Separate
None
Next
Select Model
Model Name
Simple_Nuclei_SegmentationModel
Model Class
Nucleus
OK
Cancel

Now the trained model will be used to segment the image. The built-in ZEN Tiling Client automatically to chucnk the image and deal with cmplex dimensions, like Use the **`Scenes`** etc.

Additional Porst-Processing option, incl. a Minimum Confidence Threshold can be applied to further refine the results.



Finally, the model can be loaded into ZEN by using the **Import** function on the **JSON file**.

If the model is supposed to be provided to other parties it is usually easier to exchange .czmodel files instead of SavedModel directories with corresponding JSON meta-data files.

The `czmodel` library also provides a `convert_from_json_spec` function that accepts the above mentioned JSON file and creates a CZModel:

In [11]:

```
# This ia additional way how to create a CZMODEL from a saved TF2 model on disk + JSON f
ile.
# The currently recommended way to to create the CZMODEL directly by using czmodel.conve
rt_from_model_spec
# the path to the TF2.SavedModel folder is defined in the JSON shown above

convert_from_json_spec(model_spec_path='model_spec.json',
                       output_path='model_from_json',
                       output_name = 'simple_nuclei_segmodel_from_json')
```