

DL4MIA 2021: Image Restoration

Learning Objectives

- ❑ Supervised training (CARE)
 - ❑ Run through the 3D denoising tutorial...
 - ❑ Noise2Noise training with CARE...
 - ❑ Unsupervised training (N2V)
 - ❑ Run through a N2V tutorial...
 - ❑ Denoise your own data with N2V...
 - ❑ Unsupervised training with a noise model (Probabilistic N2V)
 - ❑ Run through tutorial...
 - ❑ Bootstrapping your own noise model...
 - ❑ *Bonus:* (Hierarchical) Diversity Denoising (DivNoising)
 - ❑ *Bonus:* ZeroCostDL4Microscopy
-

Table of Contents

Exercise #0: Create a CARE conda env	2
Exercise #1: Train your first CARE network (supervised)	4
Exercise #2: What is going on behind the scenes?	7
Exercise #3: Train a CARE network "Noise2Noise"	8
Exercise #4: Train a Noise2Void network.	11
Exercise #5: Train a Probabilistic N2V network.	13
Exercise #6 (Bonus): Train and Use a DivNoising Network	16
Exercise #7 (Bonus): Use ZeroCostDL4Microscopy, e.g. for DecoNoising	17

Exercise #0: Create a CARE conda env

For the exercises below, we assume that you are already connected to the HPC using port forwarding (see exercise 00_connection).

1. Open a terminal from the jupyter server and create a new conda environment:

```
$ conda create --name care python=3.7
```

2. Activate the newly created environment

```
$ conda activate care
```

3. Now we will install all you need in this environment. For CARE (the software package is called CSBDeep) and it can be installed using another widely used package manager: *pip*. Note that the packages installed with *pip* are still installed in your current conda environment!

```
$ pip install CSBDeep
```

4. Now let's install some more dependencies:

```
$ conda install tensorflow-gpu keras jupyter tensorboard
```

5. Clone the CSBDeep repository to get the notebooks:

```
$ git clone https://github.com/CSBDeep/CSBDeep.git
```

6. Now you will be able to start playing around with the example notebooks in "CSBDeep / examples" (see next section).

Some of the notebooks in this exercise will have output cells in them. To get the full experience without spoilers, clear all outputs when first opening each notebook (in the jupyter menu navigate to `Cell > All Outputs > Clear`).

OPTIONAL BONUS (maybe come back here while one of the exercises below started to train your network): Tensorboard is a tool used to visualise the evolution of a neural network training throughout the training epochs. If you want to use it, we will need to forward yet another port to your local machine!

Just a few words about the logic of running tensorboard:

- Tensorboard needs to be started in the folder where you train a notebook.
- If you train a different notebook you will need to start tensorboard where this new notebook is.
- Tensorboard helps visualise training, you can start it during training or after.
- Similarly to Jupyter, we need to perform port forwarding from the HPC to your machine to allow you to use Tensorboard in your browser
- If you want to start multiple tensorboard, you will need to use different ports! So close your unused tensorboard processes!

1. Tensorboard typically uses port 6006, but since we are multiple users per cluster node, sometimes tensorflow will be run on the next available port. Another solution is to set a unique port per user. This is what we did with the jupyter notebook ports (<jupyter port> = something like 8889, 8890, etc. that you wrote down earlier). Jupyter notebooks usually run on port 8888, therefore you need to calculate your tensorboard port following this formula:

<tensorboard port> = <your jupyter port> - 8888 + 6006

Example: if your port in jupyter was 8891, then your port for tensorboard should be $8891 - 8888 + 6006 = 6009$.

2. Let's now forward this specific port to your local machine. There are two ways:
 - a. Either, you can close the ssh connection you established previously and run it again with an additional port forwarding flag (this is a single command):

```
$ ssh <user.name>@hpclogin.fht.org -L  
8888:<gnode>:<jupyter.port> -L 6006:<gnode>:<tensorboard.port>
```

Example: `ssh brad.pitt@hpclogin.fht.org -L 8888:gnode07:8891 -L 6006:gnode07:6007`

- b. Or you can open a new terminal window **locally** and type:

```
$ ssh <user.name>@hpclogin.fht.org -L 6006:<gnode>:<tensorboard.port>
```

Example: ssh brad.pitt@hpclogin.fht.org -L 6006:gnode07:6009

3. (LATER after you started your notebook) Once you are training a network, and if the notebook actually outputs a training log, you can start tensorboard with the following command **on the server**:

```
$ cd CSBDeep/examples/denoising3D
$ tensorboard --logdir=. --host 0.0.0.0 --port
<tensorboard.port>
```

This command starts tensorboard in the current folder. Make sure that the terminal is now in the folder where the notebooks are. Tensorboard scans the files and folder to find the logs with the information to plot. **You need to be in the care conda environment for this to work.**

4. Finally, in your browser connect to <http://localhost:6006> and explore the results. Note that this can be done while your training is progressing, allowing you to watch the “evolution” of your network in real time.

Exercise #1: Train your first CARE network (supervised)

The GIT repo 'CSBDeep' you have previously cloned into '~/DL4MIA/CSBDeep' contains multiple example notebooks:



	Name ↓	Last Modified	File size
..		seconds ago	
denoising2D_probabilistic		32 minutes ago	
denoising3D		a minute ago	
isotropic_reconstruction		32 minutes ago	
projection		32 minutes ago	
scripts		32 minutes ago	
upsampling3D		32 minutes ago	
README.md		32 minutes ago	179 B

Every example will be downloading all required training data and is itself divided into three individual notebooks that need to be executed in sequence. Maybe the `denoising3D` example is a good one to start with, but you can really pick any of these if you find another one more interesting...

Note: When opening each notebook, confirm that the correct ipython kernel is selected by checking the top right corner of the notebook, it should say: Python [conda env: care].

If not, change it in the Kernel menu: Kernel > Change Kernel > Python [conda env: care]

1_datagen.ipynb - network training usually happens on batches of smaller sized images than the ones recorded on a microscope. Hence, this notebook loads all your image data and chops it into many smaller pieces and stores it into the sub-folder 'data' (you can see that folder on the screenshot below, but likely not yet in your own example folder).

Open this notebook, read all explanations, execute all cells, ask any questions that come up - then continue below...

0 ▾ / CSBDeep / examples / denoising3D			Name ▾	Last Modified	File size
..				seconds ago	
data				34 minutes ago	
models				34 minutes ago	
1_datagen.ipynb				34 minutes ago	3.3 MB
2_training.ipynb				3 minutes ago	644 kB
3_prediction.ipynb				34 minutes ago	5.95 kB

2_training.ipynb - this notebook will train your CARE network. All outputs will be put into the folder 'models'. While you execute this notebook, see what files will be generated inside the models-folder.

As you can see, the example notebooks contain quite some additional explanations and some cells that have the purpose of showing you the data you are about to use and some sampled results. In this way you can be sure that the right things are happening.

Once you reach the cell that is actually starting the network training, you can go ahead and use Tensorboard (see the optional bonus on page 3) to monitor the training process. If you did not change the default settings you will have between 10 and 20 minutes to play with Tensorboard before the training of your first CARE network will be done. Enjoy (and please ask questions if you have any)!

At the very bottom you see that we can even export a so-called 'Tensorflow Saved Model'. Such a zip file contains all data and metadata to fully define a Tensorflow model.

3_prediction.ipynb - the last of the three notebooks can be used to apply the network we have previously trained on any dataset you'd like to. Open the notebook and see what it does.

Note: if you happen to run into weird out-of-memory errors, you will need to shutdown notebooks that occupy GPU memory but are likely not used any longer. One good way to do so is by clicking on the 'Running'-tab in Jupyter, then on the orange 'Shutdown'-button next to the notebooks that are not longer needed:

Files

Running

Clusters



Currently running Jupyter processes



Terminals ▾

There are no terminals running.

Notebooks ▾

 CSBDeep/examples/denoising3D/2_training.ipynb	Python 3	Shutdown	seconds ago
 CSBDeep/examples/denoising3D/3_prediction.ipynb	Python 3	Shutdown	seconds ago

Exercise #2: What is going on behind the scenes?

Seek to understand what happened behind the scenes while executing the three notebooks you just executed in Exercise 1.

Try to answer the following questions:

1. Where is the training and test data located?
2. How does data have to be stored so that CSBDeep will find and load it correctly?
3. Where are trained models stored? What models are stored? What is the difference between them?
4. Where in the notebooks is the name of the saved models determined?
5. How can you influence the number of training steps per epoch? What did you use (likely in the interest of time) and what value is suggested?
6. BONUS: While this is not done in the example notebooks, how would you load an existing CARE network and continue to train it?
7. BONUS: How would you do the same thing in such a way that the additionally trained model is stored separately (under a new name)?

Exercise #3: Train a CARE network “Noise2Noise”

Data by Reza Shahidi and Gaspar Jekely, Living Systems Institute, Exeter

In this exercise you will start with raw data and decide for yourself how to train a CARE network. The data contains the same sample, imaged at different levels of noise. Use CARE to improve the quality of noisy images. You will have to make decisions -- make them!

1. We are going to work with some new data. If you wish to explore the data locally, you can download it from <https://tinyurl.com/yxlqagm2>.

NOTE: since you will need this data on the HT cluster, we put it there already! You can find it in the shared folder `/group/carecourse/SEM`. Later in this exercise we will copy the data directly from there.

2. The file contains 2 *tiff-stacks*, one for training and one for testing. Open **train.tif** or **test.tif** in Fiji or with python to look at the content. Each stack contains 7 images of the same tissue that were recorded with different scan time settings of a Scanning Electron Microscope (SEM):
 - Image 0 (1 in Fiji) is recorded with 0.2 μ s scan time
 - Image 1 (2 in Fiji) is recorded with 0.5 μ s scan time
 - Image 2 (3 in Fiji) is recorded with 1 μ s scan time
 - Image 3 (4 in Fiji) is recorded with 1 μ s scan time
 - Image 4 (5 in Fiji) is recorded with 2.1 μ s scan time
 - Image 5 (6 in Fiji) is recorded with 5.0 μ s scan time
 - Image 6 (7 in Fiji) is recorded with 5.0 μ s scan time avg. of 4 images
3. Back on the HPC, make a copy of the **1_datagen.ipynb**. Rename it to **1_datagenSEM.ipynb**.
4. The data is present on the HPC server in "`/group/carecourse/SEM`", you can copy it to the **data** folder of the CSBDeep example by doing if your terminal is currently in the notebook folders:

```
$ cp -r /group/carecourse/SEM data/SEM
```

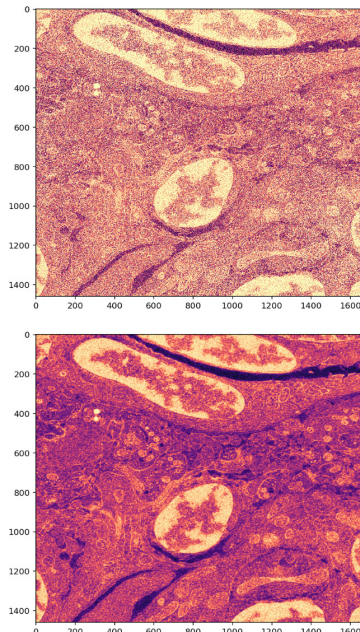
You can navigate the folders on the HPC using the `cd` command.

- How would you train a network to denoise images of 1 μ s scan time? Which images do you think could be used as input and which as target?
- Open the **training.tif** and save respective images from the stack into the **train/low** and **train/GT** folders. Use the same name for the input and target images to pair them. You can the **imwrite** function from **tiff file**:

```
In [2]: 1 from tiff file import imwrite
2 data=imread('data/SEM/train/train.tif').astype(np.float32)
3 imwrite('out.tif',data[0])
```

- Modify your **1_datagenSEM.ipynb** to work with your data and run it. Note that you are using 2D images instead of 3D stacks now.
- Make a copy of **2_training.ipynb**, modify it accordingly and train a network on your data.
- Make a copy of **3_prediction.ipynb** and modify it accordingly. Open **test.tif**, process it and look at the results for the different acquisition times.

```
In [14]: 1 # Load data, we are looking at the 1.0 usec data (slice 3)
2 Xtest=imread('data/SEM/test/test.tif').astype(np.float32)[3,:,:.newaxis]
3 restored = model.predict(Xtest, 'YXC')
4
5 #Show input, we are zooming in a bit.
6 plt.figure(figsize=(7,7))
7 plt.imshow(Xtest[1500:,:0],cmap="magma")
8 plt.show()
9
10 #Show result, we are zooming in a bit.
11 plt.figure(figsize=(7,7))
12 plt.imshow(restored[1500:,:0],cmap="magma")
13 plt.show()
```



- Can you further improve your results by using the data differently or by tweaking the settings? How could you train a single network to process all scan times? Be creative! Surprise us!

Exercise #4: Train a Noise2Void network.

Noise2Void allows training from single noisy images.

Note: you may use your own data if you have some noisy images with you...

1. Create a new folder (for example in the 'DLM4MIA' folder, in which new jupyter terminals are initiated) and clone the n2v repository:

```
$ git clone https://github.com/juglab/n2v.git
```

This repo contains all the sources that make N2V, but you clone it only to get your hands on the tutorials you will find within the `examples` folder.

2. Now create a new conda environment only for N2V.

```
$ conda create -n 'n2v' python=3.7
```

```
$ conda activate n2v
```

```
$ conda install tensorflow-gpu=2.4.1 keras=2.3.1
```

```
tensorboard nb_conda
```

```
$ pip install n2v
```

Side Notes: (i) Maybe we could have installed N2V also into the care conda environment, but not always this is possible. Some tools you will use might depend on different Tensorflow versions or different and incompatible versions of other dependencies. Therefore it is best practice to create a different environment for different projects. (ii) In addition to the installation instructions online, we also installed `tensorboard` and `nb_conda`... why? Because we love them and want to use them!!!

(iii) But... wait a sec... we didn't even install `jupyter`... what is going on? By installing `nb_conda`, `jupyter` is actually a dependency, hence, it got installed for that reason anyways.

3. Start jupyter and have a look at the **examples/2D/denoising2D_SEM/01_training.ipynb** and **examples/2D/denoising2D_SEM/02_prediction.ipynb** notebooks. Some things are different from CARE, but the overall spirit really is the same. Make sure you understand all the steps. Ask us if you don't.
4. **Note:** if you are having problems downloading the data for the training notebook in `denoising2D_SEM`, you can find a copy in the shared group

folder `/group/carecourse/n2v_data/` in here you can find a zip file (with the same name as in the previous exercise but it is actually different).

- Copy it to your local folder

```
$ cp -r /group/carecourse/n2v_data/ data/
```

5. During training (about 15 min): ask us for details you didn't fully grasp. Or start **tensorboard** and follow the learning progress! (If you forgot how, figure it out or look it up in the bonus exercise above!)
6. Replace **train.tif** and **test.tif** with your own data. Play with the parameters and have fun. (*Pro tip*: copy any of the exercise folders, name it in a sensible way, then change the contained notebooks to suit your needs...)

Power Question (think about it, we'll discuss it later):

Remember what he heard in the lecture about N2V and how it is trained by taking noisy pixels over to be used as ground-truth. What are the consequences for the loss of the network during training in terms of how we can interpret them? Any clues???

More specifically: can you think of anything that will be different in a loss plot using N2V compared to a loss plot with something like supervised CARE???

Exercise #5 (if time permits): Train a Probabilistic N2V network.

Probabilistic Noise2Void, just as N2V, allows training from single noisy images. In order to get some additional quality squeezed out of your noisy input data, PN2V employs an additional noise model which can either be measured directly at your microscope or approximated by a process called 'bootstrapping'.

Below we will give you a noise model for the first network to train and then bootstrap one, so you can apply PN2V to your own data (in this course we simply don't have the means to also do the microscopy bits to record a suitable noise model for your data...)

Note: Our PN2V implementation is written in pytorch, not Keras/TF.

Note: PN2V experienced multiple "updates" regarding noise model representations. Hence, *the original PN2V repository is not any more the one we suggest to use...* (despite it of course working just as described in the original publication...)

1. Clone our **PPN2V** repository (the one repo that can do it all):

```
$ git clone https://github.com/juglab/PPN2V.git
```

Note: In the readme on GitHub, installing PPN2V is currently suggested via a yaml file that specifies all required dependencies. This would mean to do something like (**don't do it though, go right to the next point**):

```
$ conda env create -f ppn2vEnvironment.yml
```

2. In order to get PPN2V to run, we just install stuff by hand:

```
$ conda create -n ppn2v python=3.7
```

```
$ conda activate ppn2v
```

```
$ conda install pytorch torchvision torchaudio  
cudatoolkit=11.1 -c pytorch -c nvidia
```

```
$ conda install nb_conda tifffile matplotlib scipy
```

3. Note how sneakily I made you install `nb_conda` again? Why did I not also make you install Tensorboard??? Hard question, but has a very good answer...

Ok, now that we have installed all we need for (P)PN2V, let's do something cool with it!

Remember that the difference between N2V and PN2V is that we use a noise model? Remember that we can record data at the microscope that allows us to get such a noise model? We don't have the equipment (or time) during the course to do that, but we can run through an example that comes with all required noise model data:

1. Run through the tutorial in 'examples/Convallaria/PN2V'.
2. Note a few things:
 - a. There are also folders for CARE and N2V... why the heck would that be? Spoiler: (P)PN2V is the first method we use that is implemented in pytorch instead of Tensorflow and we implemented our own baseline methods also using pytorch so we can compare performances more directly.
 - b. Inside the PN2V folder, you find two notebooks for step 1. The first one, 1a, uses recorded images as calibration data to compute a noise model. The second one, 1b, is the one to use if you want to bootstrap a noise model (aka, denoising your data with N2V, then using the noise raw data and the N2V denoised images to compute a noise model). **We suggest you start with 1a!**
 - c. In all the noise model generation notebooks you will find code to create a histogram-based noise model and a GMM based noise model. Strictly speaking, you don't need to create both, so... make your choice!
 - d. You might wanna reduce the number of epochs before you start the training in order to speed things up.
3. Now try PN2V on some of your own data! Oh... and... yes... very likely you don't have calibration data and need to bootstrap a noise model. For this you can either use N2V from the conda env you installed in the previous exercise, or you use the pytorch N2V version we found just before in the PPN2V repo (this might be simpler since the bootstrapping notebook expects this to have happened)... good luck! Ask us in case of trouble!!!

Exercise #6 (Bonus): Train and Use a DivNoising Network

DivNoising is one of the latest unsupervised denoising methods and follows a quite different approach. Instead of a U-Net, DivNoising employs the power of a Variational Auto-Encoder (VAE), but adds the use of a noise model to it in a suitable way.

This approach comes with an extra perk: you will be able to sample diverse interpretations of a noisy image. Why is that cool, you ask? Easy:

- If the diverse samples look all the same (or VERY similar to each other), you know that the noisy data you just denoised is not very ambiguous and you might decide to trust the result more.
- If, on the other hand, the samples look all quite different, you now know that you might not want to trust any of the denoised 'interpretations'. Additionally we show in the DivNoising paper how diverse samples can be used in meaningful ways for improved downstream analysis of your data.

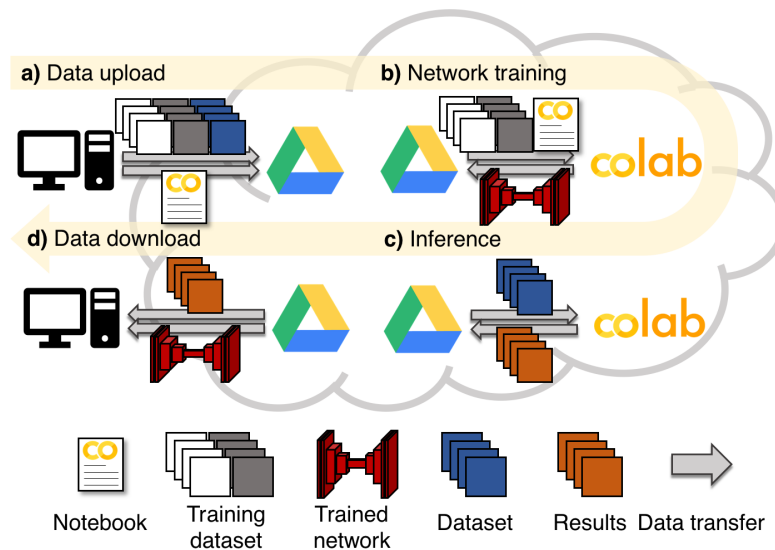
Since this is a bonus exercise and you are a pro by now, we will keep the instructions here brief, essentially putting you in the position you would find yourself in when you come across a method you find interesting and want to check it out:

1. Find DivNoising here: <https://github.com/juglab/DivNoising>
2. Follow the installation instructions you find in the readme on GitHub.
 - a. Yes, the author of DivNoising and the readme installs things slightly differently than we are used to by now... just follow his advice, you will be fine...
 - b. Now start jupyter, just as we did in all the exercises before...
3. Follow the good advice in the readme and maybe run through the Convallaria example?

Side note: DivNoising is based on pytorch, but it is using a library called PyTorch Lightning. Why should you care? I tell you: you can use Tensorboard to follow the training progress of your DivNoising network! Yay!

Exercise #7 (Bonus): Use ZeroCostDL4Microscopy, e.g. for DecoNoising

This exercise sheet will introduce you to [ZeroCostDL4Mic](#), a toolbox for the training and implementation of common Deep Learning approaches to microscopy imaging. It exploits the ease-of-use and access to GPU provided by [Google Colab](#).



Learning Objectives

- ☐ Familiarize yourself with Google Colab, and how to use it.
 - ☐ Familiarize yourself with what ZeroCostDL4Mic (aka 'Zero') has to offer.
 - ☐ Denoise microscopy images using Noise2Void
 - ☐ [BONUS] Denoise microscopy images using DecoNoising
-

The Google Colaboratory, or "Google Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a student, a data scientist or an AI researcher, Colab can make your work easier. If you're really lost, you might watch [Introduction to Colab](#) to learn more, or just go on reading and working through this exercise sheet.

If you have never ever used Google Colab, please take a few minutes to quickly go over the following notebook (provided from Google for this purpose exactly). <https://colab.research.google.com/notebooks/intro.ipynb>

So... having access to GPUs from everywhere? Sounds like a dream! This was also what the [people behind Zero](#) thought when they started collecting useful deep learning ideas and casting them into usable notebooks that run on the Google Colab.

A huge advantage is not only that it is free, but all these different models can now be trained using a very similar workflow. Once you go through any of the many Zero notebooks, the others will look similar and take even less time for you to run.

Now... browse through the [long list of notebooks](#) that are available today. You will spot many 'old friends', such as N2V, CARE, etc. Please also note that new notebooks are added quite frequently, so next time you check this page, you might find more useful functionality you might benefit from in your daily work.

Important steps before you are good to go:

- You will need a Google account. If you don't have one, today might be the day to [get one](#)... (alternatively, you might now focus on the EmbedSeg exercise sheet... ;)
- For Google Colab to make the most sense, it is by far the most convenient to store all data on [your Google Drive](#). (It's not strictly required, but since the alternative is so annoying, we will assume you know how to upload and download data to your Google Drive.

Run DecoNoising in Zero:

As mentioned in the lecture, DecoNoising is a fun little idea on top of Noise2Void. (Remember, we introduced a fixed convolution layer just therefore

the network output, forcing the deep net to learn some sort of deconvolved image in the layer before that fixed convolution.)

When you look for [DecoNoising on the Zero Wiki](#), you will notice that it is currently under Beta testing. This means two things: (i) the notebooks are likely to change a bit over the next weeks or months, and (ii) there might be some glitches or smaller issues to encounter when using it. This being said, we used it ourselves many times and it works very well already.

Now, start the DecoNoising notebook from the [Zero page](#), or directly from [here](#).

Important Note: for the DecoNoising notebook to work, you need at least two training images and all training images need to have the same size in pixels. Hence, if you want, for example, to use the SEM images from one of the previous exercises, you will first have to cut them into smaller images that have the same dimension. If you are lazy, which you really should, feel free to download such crops from [here](#).

Bonus Bonus Exercise: if you decide to also store the 'deconvolved' network output (aka the activation of the network layer just before the fixed convolution layer), try opening that TIFF file in Fiji (it will fail, since it is a 64bit TIFF which Fiji does not understand). You can currently only 'fix' this by changing the code in the DecoNoising notebook a bit. If you feel like spending some time... try to fix this! (Spoiler: you need to change the datatype to be e.g. float32...)