# Data importing and modification

All data used is in the form of DICOM files locally saved in a data directory. To load three dimensional arrays of the structures and the dose, the data\_import.py script is used. Requires NKI AVS modules to work. To do data augmentation on the training data, the data\_augmentation.py script is used.

data\_import.py

**Summary:**Gives the structure mask, the dose array and the number of voxels added for the standardized size on each side of the array.

**Other scripts used:**- none

**Data used:**DICOM data

**First time modifications:**Modify the first part of the input\_data function. Modify the file locations of the patient list, structure list and PTV list. Apart from that modify the data directory where the DICOM files are located. Finally modify the preferred input size.

**Usage:**import data\_import

structure, dose, startmod, endmod = data\_import.input\_data(Npat)

**Input & output:**input\_data:  
IN: Npat: Number of the patient (0 to N)  
OUT: structure: Array with structure masks  
 dose: Array with dose  
 startmod: Array with number of added voxels at start of array  
 endmod: Array with number of added voxels at end of array

data\_augmentation.py

**Summary:**Outputs an augmented Torch Tensor based on the transformation parameters used as input, together with the structure or dose array to be modified.

**Other scripts used:**- none

**Data used:**No new data used

**First time modifications:**Nothing to modify

**Usage:**import data\_augmentation as aug

trans\_list = aug.trans\_list()

str\_tens = aug.structure\_transform(structure, tr\_val)

dos\_tens = aug.dose\_transform(dose, tr\_val)

**Input & output:**trans\_list:  
IN: -  
OUT: trans\_list: Predifined list of transformations, first 3 indices transformation per axis. Last index either 0 for translation or 1 for rotation  
  
structure\_transform:  
IN: structure: Array with structure masks  
 tr\_val: transformation values  
OUT: str\_tens: Torch tensor with augmented structure masks

dose\_transform:  
IN: dose: Array with dose values   
 tr\_val: transformation values (same as structure transform)  
OUT: dos\_tens: Torch tensor with augmented dose values

# Dose prediction neural network training

Neural network training based on structures as input is done using the models defined in U\_Net.py. The Model\_exec.py script is the script that is used to execute the model training. The structure based neural network prediction works in the same way as the segment prediction network and therefore has similar usage.

model\_exec.py

**Summary:**Script that is the basis for executing neural network training.

**Other scripts used:**- data\_import.py  
- data\_augmentation.py  
- U\_Net.py

**Data used:**  
Predicted/correct segment dose when doing the multi stage training.

**First time modifications:**Make sure that the paths to the pat\_list and the dat\_folder are correctly defined.

**Usage:**  
Before starting training, select the correct model in the line model = UNet(). For the structure based prediction U-Net UNet(), the only predicted dose model InDoseUNet(), or the combined approach prediction SeqUNet().

Next, select the proper input for the chosen model in both the training and the validation loop. This is done by uncommenting the structure\_1 or structure\_6 line or keeping them both commented for only the structure as input. Which is chosen depends on the model choice.

Next choose the loss function. Either the standard MSE loss, or one of the custom losses heaviweight or MSE weight. When choosing a custom loss, uncomment the loss line containing the custom loss function and the structure as input. When choosing the MSE loss, uncomment the loss\_func()line. This needs to be done in both the training and the validation loop.

Finally, it needs to be defined whether to use augmentations or not. To include augmentations in training use aug\_list = trans\_list. When not using augmentations in training use aug\_list = trans\_val\_list.

After setting all this, the script can be ran entirely and will run for the amount of epochs specified.

**Loading and saving models:**

The network parameters are saved in .npy files for every model individually. To save the parameters of a model to npy files, the following lines of commands are used:

torch.save({

‘epoch’: epoch,  
‘model\_state\_dict’: model.state\_dict(),  
‘optimizer\_state\_dict’: optimizer.state\_dict(),  
‘loss’: loss,  
}, ‘Path/to/model’

)

To load the parameters of the model from the npy files, first the model needs to be defined by running the model = UNet() line. Next the parameters are loaded into the model by the following commands:

checkpoint = torch.load(‘Path/to/model’)  
model.load\_state\_dict(checkpoint[‘model\_state\_dict’])  
for state in optimizer.state.values():  
 for k, v in state.items():  
 if isinstance(v, torch.Tensor):  
 state[k] = v.cuda()

# Dose engine execution

The dose engine is executed using the dose\_eng\_exec.py script. To be executed, it needs the DE\_functions.py script for several dose calculation parts. For extraction of segment MLC contours the Segment\_extr.py script is used. Finally to use a predicted segment a post processing needs to be done with DE\_post\_pros.py. As a final note, the script now retrieves information such as the isocenter position from the RT\_PLAN. This is for convenience and should be retrieved from other sources when the plan is not known.

dose\_engine\_exec.py

**Summary:**Script executing the dose engine calculation. It works by creating a matplotlib path of the MLC contour and calculating the dose distribution using the MLC path and various functions.

**Other scripts used:**- data\_import.py  
- Segment\_extr  
- DE\_functions  
- DE\_post\_pros

**First time modifications:**Make sure that the paths to the pat\_list, patIDs and the dat\_dir are correctly defined. Also correctly define the paths to the data in DE\_post\_pros.py and the path to the kernel information in DE\_functions.py.

**Usage:**Choose whether to use the prediction using the correct segments or the predicted segments. Uncomment one of the corresponding lines:

out, rel\_weight = pp.post\_pros(beam\*70+cp, Npat)

out = Beampath(plan\_dicom.BeamSequence[beam], cp)

The first line for the predicted segments, using the post processing script. The second line for the correct segments.

The output is generated in the lines:

hit\_checked\_arr = DE\_functions.hitchecker(input)

conv\_out += DE\_functions.ccconv(input)

Where the first line calculates the TERMA values and the second line calculates the collapsed cone approximation using the dose kernels.

DE\_functions.py

**Summary:**Contains all functions corresponding to the TERMA and collapsed cone approximation calculation.

**Usage**For the calculation of the TERMA of a single beam direction:

hit\_checked\_arr = DE\_functions.hitchecker(input)

For the calculation of the collapsed cone approximation of a single beam direction:

conv\_out = DE\_functions.ccconv(input)

**Input & output:**hitchecker:  
IN: loc\_array: array with voxel locations with respect to the isocenter  
 SAD: distance from source to isocenter  
 ang: beamangle   
 MLCpath: matplotlib path of MLC contour  
 ext: Boolean map of the body  
 pos: position of first voxel  
 isocenter: position of the isocenter  
OUT: out\_arr: 3D array with the TERMA calculated voxel values  
  
  
ccconv:  
IN: rel\_out: Relatively weighted TERMA values for voxels  
 ext: Boolean map of the body  
 Sourceloc: location of the source  
OUT: conv\_out: Output of collapsed cone approximation.

DE\_post\_pros.py

**Summary:**Calculates the MLC contour from a segment prediction. Needs the information of the contour prediction. Only use the postprocessing step when using segment predictions.

**First time modifications:**Make sure to correctly define the paths to the data in DE\_post\_pros.py. The data consist of MLC projections for all control points on the arc in the third dimension as predicted in the segment prediction part. Also, the correct value in the line test = inp[slice+2] > N needs to be defined.

**Usage**Calculate the output and relative weight from the CP and patient number:

out, rel\_weight = pp.post\_pros(N\_CP, Npat)

**Input & output**post\_pros:  
IN: N\_CP: Beam control point  
 Npat: patient number  
OUT: out: Array that can be converted to matplotlib path.

Segment\_extr.py

**Summary:**Extracts MLC position information from the RTPLAN instead of predicted values.

**Usage**Calculate the output and relative weight from the CP and patient number:

out = Beampath(DICOM\_info, N\_CP)

**Input & output**Beampath:  
IN: DICOM\_info: Beam sequence information.  
 N\_CP: Beam control point  
OUT: out: Array that can be converted to matplotlib path.

# Segment prediction networks

These scripts are used to predict MLC contour segments with

Seg\_exec.py

**Summary:**Script that is the basis for executing neural network training. It works the same as the dose prediction network, with the exception of loading different data and using different models defined in segment\_unet.py.

**Other scripts used:**- data\_import.py  
- data\_augmentation.py  
- Segment\_unet.py

**Data used:**  
Structure projections from BEV which can be created using MLC\_pred.py. Apart from that, the true contour and the weights are used to train the network on.

**Usage:**  
Overall the same usage as the dose prediction script. Need to set the correct loss function by uncommenting the

After setting all this, the script can be ran entirely and will run for the amount of epochs specified.

**Loading and saving models:**Same as in dose prediction networks.

MLC\_pred.py

**Summary:**Script containing functions to create input values for the segment prediction neural network.

**Usage:**trans\_maps(patient) function is called to calculate the input data for the neural network. The real\_seg\_cont(patient) function is called for the creation of the true contour. Furthermore, the outcome needs to be saved to use it again within the neural network.

# Other scripts

Several scripts that are used for evaluation and other purposes.

OVH\_pred.py

**Summary:**Script that calculates the rectum DVH based on a an OVH prediction

**Usage:**First , the correct file locations need to be defined of dfOVH, dvhFile and the training patients. Next call resultDVH = OVH\_pred.OVHpred(patID) to make the prediction.

Dose\_char.py

**Summary:**Script which contains several functions that calculate individual dose characteristics.

**Usage:**Call individual functions for the extraction of different dose statistics.

Model\_val.py

**Summary:**Script containing a chaos of all kinds of different things to test the outcome of the dose prediction neural network.

**Usage:**trans\_maps(patient) function is called to calculate the input data for the neural network. The real\_seg\_cont(patient) function is called for the creation of the true contour. Furthermore, the outcome needs to be saved to use it again within the neural network.

Plotting.py

**Summary:**Contains various predefined plots, such as a loss plot of the training, a dose difference plot, a DVH plot, a DICE plot and a contour plot.

**Usage:**Call individual functions for different plots.