DELFT UNIVERSITY OF TECHNOLOGY

BSC APPLIED PHYSICS

BACHELOR THESIS

The application of Deep Learning to improving low count SPECT imaging

Supervisors:
Dr. ir. M.C. Goorden
ir. M.P. Nguyen

Author:
C.K. den Ouden (4563913)

Thesis Committee:
Dr. ir. M.C. Goorden
Dr. Z. Perkó

June 16, 2020

Abstract

Preclinical SPECT systems such as the U-SPECT have been able to achieve sub-half-millimetre spatial resolution with the use of cylindrical pinhole collimators. Utilising this type of collimator comes at the cost of a reduction in the total number of detection events that take place. In order to compensate for this either the activity of the radiopharmaceutical must be increased or the exposure time must be extended. Ideally the dose received by a patient or subject is kept to a minimum. The goal of this study was therefore to investigate the application of deep learning to SPECT imaging, specifically to improve low count images to resemble high count images in the projection domain. The projection domain was chosen over the image domain as projections are easily generated in large quantities, while reconstructed images take large amounts of time and computation to generate.

A previous BSc Thesis study constructed a neural network (a Perceptual Loss Network) to this end, and it was concluded that the training set was too small and specific for the neural network to be more generally applicable. In this study therefore the training set of the neural network was expanded on with different phantom types such as Derenzo hot rod phantoms, Jaszczak phantoms and uniform phantoms of various shapes and sizes. Phantoms were simulated and measured using the EXIRAD-3D, and projection pairs (low and high count) were generated. Several network architecture improvements were also explored, such as processing the projection images from different detectors together using width concatenation or applying downsampling.

Phantom test sets were used in order to determine whether expanding the training set and making adjustments such as width concatenation or applying downsampling had a positive effect on the neural network's ability to improve varying SPECT projections. The projections of these test sets were improved by the neural networks and then reconstructed to 3D arrays in the image domain. The reconstructions would then be compared against the low count reconstruction as well as those produced by the original neural network. It became apparent that the neural networks do not perform well on very low count projections. It is assumed that this is because the projections have too little information contained within them for the neural networks to determine how to improve them. It may be possible to improve the neural networks by expanding the training set further with very low count projections.

The quality of the reconstructions was determined quantitatively using the Contrast to Noise Ratio (CNR) for Derenzo-type phantoms and uniformity for uniform type phantoms. Expanding the training set showed slight improvement in reconstruction CNR but was not considered significantly better. Applying width concatenation as well as expanding the training set seemed to improve results further, but the increase in resource requirements and computing time may not be justified for the marginal increase in CNR. Expanding the training set and applying downsampling proved to be very promising, increasing the CNR from anywhere between 0.35 to upwards of 0.75 in some cases. It also showed the most potential when it came to improving physically measured SPECT projections. The uniform phantoms had varying results. Very low count uniform cylinder phantoms were able to be improved by neural networks, but did not seem to benefit much from training on the expanded training set. It also seemed that trying to improve higher count uniform cylinder phantoms was difficult for the neural networks as there seem to be artifacts in the neural network reconstructions that decrease uniformity. It is recommended to further examine and improve the downsampling technique used in this study. It may also be interesting to combine the individual improvements, expanding the training set, using width concatenation and applying downsampling simultaneously, to see whether this can offer a better neural network for improving low count SPECT projections.

Contents

Abstract ii

1 Introduction 1

2 Theory 2
- 2.1 SPECT System 2
- 2.2 Imaging & Reconstruction 3
- 2.3 Neural Network 4

3 Methods 6
- 3.1 PLN Expanded 7
- 3.2 PLN Expanded + Concatenation 8
  - 3.2.1 Width Concatenation 8
  - 3.2.2 Channel Concatenation 9
- 3.3 PLN Expanded + Downsampling 10
- 3.4 Evaluation 11

4 Results 13
- 4.1 Simulated Derenzo Phantom 1 13
- 4.2 Simulated Derenzo Phantom 2 14
- 4.3 Simulated Derenzo Phantom 3 15
- 4.4 Measured Derenzo Phantom 16
- 4.5 Simulated Uniform Cylinder 1 17
- 4.6 Simulated Uniform Cylinder 2 18
- 4.7 Measured Uniform Cylinder 19

5 Discussion 20
- 5.1 A note on image formatting 21

6 Conclusion 22

References 23

Appendix 25
- A Training set 25
  - A.1 Derenzo Phantom (6 segments) 25
  - A.2 Jaszczak Phantom 25
  - A.3 Uniform Cylinder Phantom 26
  - A.4 Uniform Ellipsoid Phantom + Background 26
  - A.5 Derenzo Phantom (5 segments) + Background 27
- B Test sets 28
  - B.1 Simulated Derenzo Phantom 1 28
  - B.2 Simulated Derenzo Phantom 2 28
  - B.3 Simulated Derenzo Phantom 3 29
  - B.4 Measured Derenzo Phantom 29
  - B.5 Simulated Uniform Cylinder 1 30
  - B.6 Simulated Uniform Cylinder 2 30
  - B.7 Measured Uniform Cylinder 31
- C Jupyter Notebook of PLN Expanded 32

1 Introduction

The field of machine learning has seen significant advancements over the past few years, particularly in the area of deep learning. As faster and more powerful computer components are becoming both more affordable and more commonplace this opens up the application of deep learning to other fields as well. Biomedical imaging is one of the early adopters of machine learning, and has been used at least as far back as 1995 [1]. This research will focus on the application of deep learning, a subsection within the field of machine learning, to Single-Photon Emission Computed Tomography (SPECT). SPECT systems offer a way of obtaining 3D images by injecting radiopharmaceuticals, and then measuring the photons emitted by those radiopharmaceuticals with scintillation cameras. While clinical SPECT systems typically have a resolution of around 12mm to 8mm [2], preclinical SPECT systems have been able to achieve sub-half-millimetre resolution [3].

As SPECT systems require the injection of radioisotopes it is important to keep the dose the patient or subject receives to a minimum, a safety concept widely known as ALARA (as low as reasonably achievable). This can generally be achieved in two ways; either by reducing the activity level of the radiopharmaceutical or by reducing the exposure time (e.g. by picking a radiopharmaceutical with a short half-life). While this is beneficial to the patient's or subject's health, one major downside is that the images projected by the SPECT system will be low-count, meaning that the signal-to-noise ratio (SNR) is going to be low as well. As a result the image reconstructed from these projections will be of a lower spatial resolution. Ideally we would like the SPECT system to produce high-count projections while also keeping the dose the patient or subject receives to a minimum. This is where deep learning may be able to help.

Deep learning is a machine learning method based around creating Artificial Neural Networks (ANNs). It is sometimes considered to be too much of a black box. There would be no oversight into, or control over, how the neural network is trained. Yet this is exactly what makes deep learning such a powerful tool. Instead of having to manually tweak each parameter, each weight, one can allow the network to tweak its own weights with the goal of minimising some user-defined loss function, a process simply called training. Deep learning has a tremendous number of wildly varying applications across many fields, from enabling self-driving cars to detect obstacles on the road [4], to improving virtual assistants for smart environments [5], to performing image super-resolution [6]. The latter of these is an example of an image-to-image type neural network, a network type well-suited for this study.

This study was performed under the Biomedical Imaging group, a section within the department of Radiation Science & Technology in the Applied Sciences faculty of the Delft University of Technology. The Biomedical Imaging group develops high-resolution preclinical SPECT systems. One of the very high resolution scanners developed by them is a pinhole-based system called U-SPECT [7, 8], commercially available from MILabs BV [9].

The goal of this study was to investigate how to enhance a previously established Perceptual Loss Network (PLN) such that it could improve low count SPECT projections to more closely resemble high count SPECT projections in order to obtain a higher quality reconstructed image. One could choose to train a neural network to improve the reconstructed images directly, but as image reconstruction is very time-consuming it would be difficult to create a large training set for the neural network. It was therefore deemed best to try and improve the projection domain instead, for which it is very easy to generate a large amount of projections relatively quickly.

Both simulated and physically measured SPECT projections were used in this study. The projections were simulated and measured using the EXIRAD-3D, a preclinical U-SPECT system for micron-resolution ex-vivo imaging. The experiment was conducted using Python 3 [10] with the help of Jupyter Notebooks [11], and the fast.ai library [12] was used to provide the required neural network architectures. The free cloud GPU rental service Paperspace [13] was also used to provide access to better GPUs when needed. A previous BSc Thesis study by A.M. Visser (citation currently unavailable) into the subject was primarily focused on determining what kind of neural network would produce results closest to the ground truth. This study on the other hand focused on expanding the training data in the hopes of creating a more generalised model that can produce predictions for many different kinds of projections. The neural network used here was a Perceptual Loss Network (PLN) [14] using a ResNet-34 model pretrained on ImageNet.

Section 2 will describe the workings of the SPECT system used in this study, both how projections are generated and how they are reconstructed, as well as the inner workings of a neural network. Section 3 will be used to go into detail about the exact neural networks used, which training data was used, how they operate and which improvements were explored. In section 4 the results of training the network and reconstructing the image will be shown, which will then be discussed in section 5. Lastly, conclusions will be drawn in section 6 on how the neural networks performed, which tactics worked and which did not, and where to go from here.

2 Theory

The process of improving low-dose SPECT projections is something that spans both Physics and Computer Science, and thus it is useful to provide some background on and explanation for both fields. The first subsection will explain SPECT systems in order to provide understanding into what projections are, how they are created and how they are turned into useful 3D images. The second subsection will go into detail about the theory behind neural networks and how they operate.

2.1 SPECT System

As mentioned before, the SPECT system used in this study is the EXIRAD-3D developed by MILabs [9]. The EXIRAD-3D offers a SPECT option that uses U-SPECT-II design: a preclinical Ultra-high-resolution SPECT [7, 8]. Its design is displayed below in figure 1.

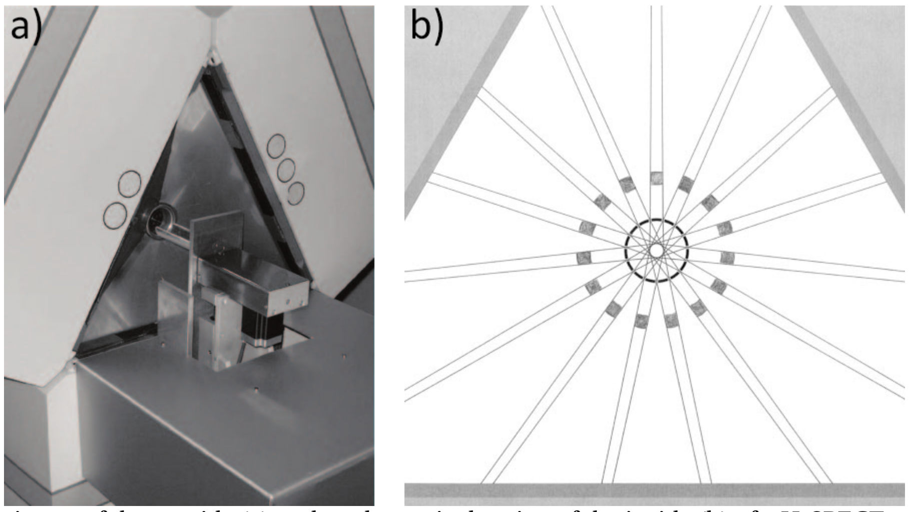


Figure 1: A picture of the outside (a) and a schematic drawing of the inside (b) of a U-SPECT system [7, 8].

The design of preclinical SPECT systems such as the U-SPECT differs somewhat from conventional (clinical) SPECT systems. A U-SPECT system utilises three scintillation detectors in a triangular arrangement. Each detector consists of a gamma camera with 10% FWHM energy resolution at 140keV and a NaI(Tl) scintillation crystal of 497.4mm by 410.6mm [15]. The collimators that would normally be attached to each of the detector plates have been replaced with one cylindrical collimator like the one in figure 2a. The collimator used in the EXIRAD-3D is of similar design to this but has slightly different specifications. The inner radius of the cylinder is 10.5mm, while the outer radius is 18mm. The collimator contains 87 gold pinholes, with their centres at 12.5mm from the central axis of the collimator. Gold was chosen as pinhole material as it has a high photon-stopping power [16], thus reducing photon penetration and scattering. The main body of the collimator is made out of an alloy of tungsten (92.5%), nickel (5.25%) and iron (2.25%). Lead shielding of 15mm thickness was placed around the collimator to prevent photons from leaving the collimator anywhere but through the pinholes. This shielding causes the well-defined edges for each pinhole projection that can be seen in figure 2b. When a photon does fall through a pinhole it ends up on one of the three scintillation detector plates after which the event is stored as a single "count" together with details like when the event took place, on which detector plate and which location the event took place and which energy was measured.

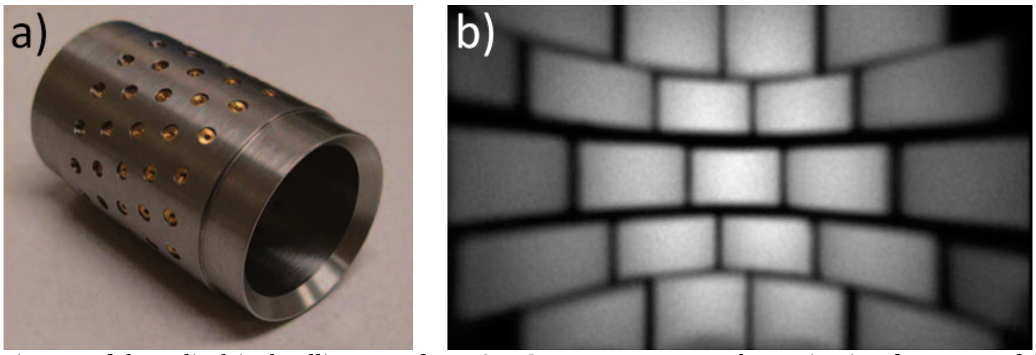


Figure 2: An image of the cylindrical collimator of a U-SPECT-II system (a) and a projection from one of the detector plates imaging a bottle of ^{99m}Tc (b). [7]

The cylindrical collimator with its pinholes of only 0.6mm diameter allows for higher resolutions than regular parallel-hole SPECT systems due to its large magnification factor. There is however a major downside to using a cylindrical collimator with a very high spatial resolution. Even though the diameter of the collimator is around 21mm, the central field of view (i.e. the area that can be seen by most or all pinholes) is only around 4mm in diameter. In order to scan something that is larger than this, the bed position must be changed. The bed is what the object to be scanned is resting on while in the device and can be moved to different positions such that eventually all parts of the object have been in the centre field of view at least once.

2.2 Imaging & Reconstruction

In order to get a sense of how well a SPECT system performs phantoms such as the one in figure 3 can be utilised. Derenzo hot rod phantoms consist of several capillary or rod segments of varying diameters which can be filled with radioisotopes. The rods within each segment have identical diameters and each rod is separated from edge to edge by that same diameter. This makes Derenzo phantoms well-suited for determining the spatial resolution of SPECT systems.

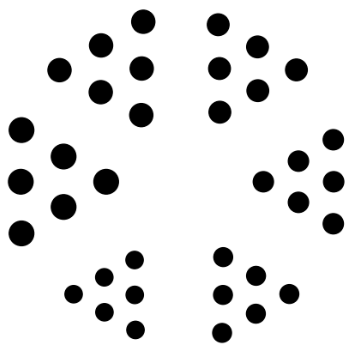


Figure 3: A schematic drawing of a transaxial cross section of a Derenzo hot rod phantom.

Ideally all projections required for this study would be physically measured, but as that both brings safety risks with it and would be quite time-consuming all training set projections used will be simulated. The SPECT system will be simulated using Geant4 Application for Emission Tomography (GATE) V8.0, a Monte Carlo simulation platform [17, 18, 19]. The system setup will be identical to the EXIRAD-3D discussed above. Scatter is not considered in the simulations as the neural network should first be able to improve projections without scatter included. The exact phantoms that were simulated will be detailed in section 3 and Appendix section A.

A 3D image detailing the activity density can be obtained by performing a reconstruction on a complete set of projections. A complete set of projections consists of three projections (one for each detector plate) per bed position. This means for example that a simulation conducted with nine bed positions requires $3 \cdot 9 = 27$ projections to form a complete projection set. The three detector plate projections belonging to the same bed position are then concatenated together to form a rectangular image file, resulting in 9 of these files. Figure 4 displays an example concatenated image file of a Derenzo phantom.

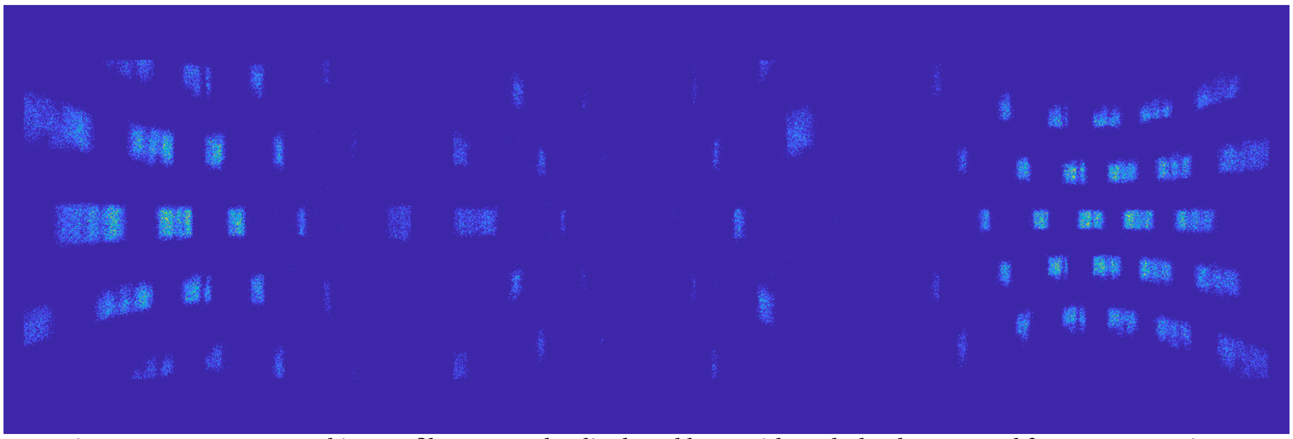


Figure 4: A concatenated image file (grayscale, displayed here with scaled colours) used for reconstruction.

The reconstructions in this study were performed using the SROSEM (Similarity-Regulated Ordered Subsets Expectation Maximisation) algorithm [20]. This algorithm has improved speed over the widely used OSEM algorithm [21] as it automatically reduces the number of subsets in regions with low estimated activity, while also avoiding artifacts such as disappearing low activity structures due to a high number of subsets. The reconstruction was performed using a method developed by F. van der Have [22]. The maximum number of subsets used was set to 128, and up to 40 iterations were performed. Once the reconstructions had finished they were run through an automated MATLAB [23] script that evaluates for every iteration which Gaussian filter results in the highest Contrast-to-Noise Ratio (CNR, see section 3.4 for details) by cycling through a list of standard deviations that define the filter size. The optimal iteration and accompanying standard deviation can then give us the best possible reconstruction in the set.

2.3 Neural Network

The most important part of the process of improving low-dose SPECT projections is the neural network. A neural network that can perform image-to-image tasks is required and thus a U-Net [24] will be used. A U-Net is a Convolutional Neural Network (CNN) that was originally designed to specifically perform biomedical image segmentation. A CNN works by repeatedly applying convolution operations on some data, in this case a projection image, in order to extract feature maps from it. A simple example of such a convolution can be seen in figure 5 and figure 6.

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter / Kernel

1	0	1
0	1	0
1	0	1

Figure 5: An example input image and kernel. [25]

Input

1	1	1	0	0
0	1	1	1	0
0	0	1x1	1x0	1x1
0	0	1x0	1x1	0x0
0	1	1x1	0x0	0x1

Feature Map

4	3	4
2	4	3
2	3	4

Figure 6: A feature map has been created after convolution of the kernel with the input image. [25]

Figure 5 displays some example input data of size 5x5 and an example kernel of size 3x3. Convolving the two will result in something like figure 6. This convolution takes the sum of the products of where the kernel overlaps with the input. In this case, the kernel was moved over one space to the right each time step. This means that what we call the *stride* of the kernel is 1. Additionally no *padding*, that is to say a border of zeroes around the input, was applied during convolution. Together the kernel size, stride and padding determine the dimensions of the feature map. Note that the size of the feature map is now 3x3, smaller than the original input of 5x5. By varying these parameters multiple feature maps can be obtained from the same input data, each with slightly different features extracted from it. By repeatedly applying these convolutions the network can form more and more layers. Figure 7 displays the general shape of such a CNN.

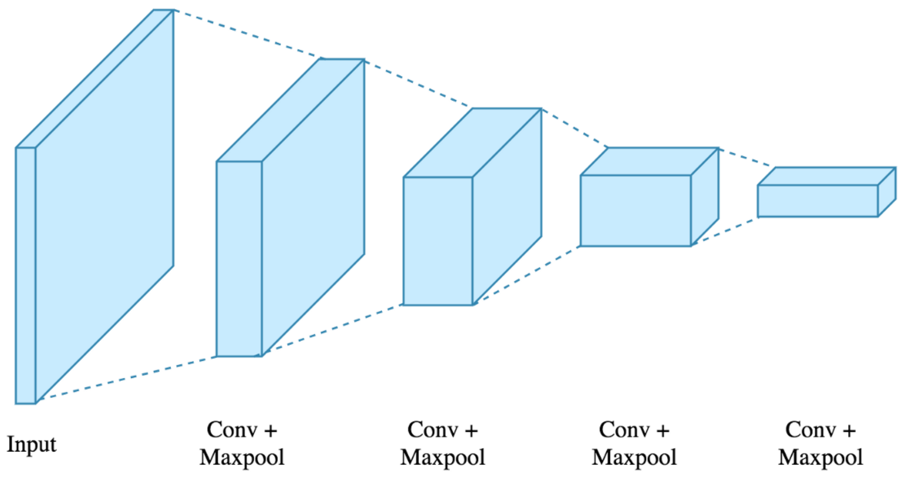


Figure 7: An example of a CNN structure. As the input resolution becomes smaller with each layer, the number of feature maps grows larger. [25]

It is good to remember that these convolutions often deal with data that is 3D or more. For example, an RGB image of 512x512 is not stored as a 2D array but as a 3D array of size 3x512x512, where the extra dimension is used to store each colour channel separately. The kernel will in this case also be 3D rather than 2D, and the resulting array of feature maps will be 4D, i.e. *n*x3x512x512, where *n* is the number of feature maps.

A U-Net consists of two parts: a contracting part and an expanding part. These two parts are what gives the network its distinct U-shape, as can be seen in figure 8.

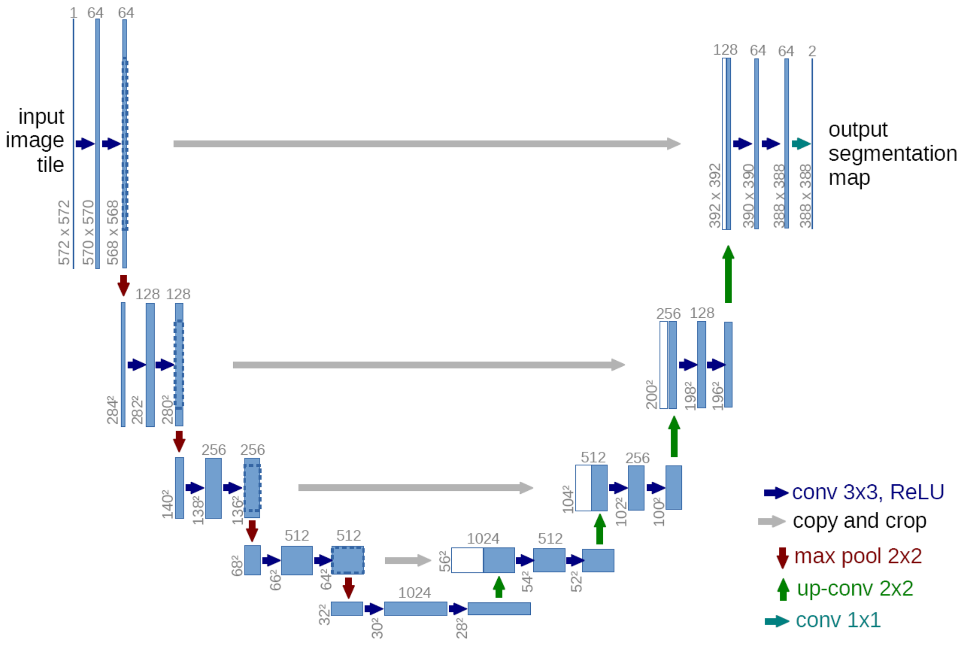


Figure 8: The well-known U-Net architecture. The number of feature maps is displayed at the top of the blue boxes. The dimensions of each feature map is displayed to the bottom left of the blue boxes. The differently coloured arrows denote the various operations. [24]

The contracting part is in essence a CNN, consisting of several convolutions, Rectified Linear Units (ReLUs) and max pooling operations. A ReLU ensures that any values that accidentally end up negative after convolution are set to zero and is defined as

$$ReLU(x) = max(0, x) \tag{1}$$

where x is iterated through every value of the array. Meanwhile max pooling operations are somewhat similar to convolution operations. In the case of figure 8 max pooling will examine each feature map in 2x2 chunks and take the maximum value in each chunk as the new value. This causes both the feature map height and width to halve. Max pooling is mainly done to extract only the sharpest features, while also keeping down the number of computations. The expanding part of the U-Net is very similar in structure to a regular CNN, except now each convolution decreases the number of feature maps while increasing the image resolution. Figure 8 also contains a few "copy and crop" connections that link very distant parts of the network together. These connections are more commonly called skip connections and form the basis of ResBlocks and ResNets [26] which allow networks to be much deeper, i.e. have more layers, than was possible before these came about.

The goal of any image-to-image neural network is generally to minimise some predefined loss function. This is achieved by training the network on a dataset of image pairs. Each image pair consists of a low quality and high quality version of the same image. The neural network can then try predicting what the high quality version would look like based on the low quality image. The simplest and most obvious loss function for image-to-image tasks is to calculate the Mean Squared Error (MSE) between the prediction made by the neural network and the real high quality image, which is often called the ground truth. Based on this loss the network can tweak the weights stored in its layers a very small amount (values around 10^{-3} and 10^{-4} are common) to try and minimise the loss for the next image pair. By training the neural network on a large number of image pairs the network should eventually converge to the global minimal loss. There is however always a risk of overfitting where a model becomes very good at handling the particular dataset it was trained on, but performs poorly on similar but new data. This problem will hopefully be avoided in this study by expanding the training set with a variety of phantoms.

While an MSE loss function is common there are others that may prove more useful for the task at hand. During a previous BSc Thesis study it was determined that a Perceptual Loss Network (PLN) [14] proved most suited for the desired image-to-image task. An MSE loss function only looks at the final prediction of the neural network and compares that to the ground truth, while a perceptual loss function compares the outputs of each convolutional layer between the predicted image and the ground truth. This is why perceptual loss is also sometimes called feature loss.

3 Methods

One of the aims of this study is to enhance the previously established PLN by increasing the training dataset. Previous BSc Thesis research made it apparent that although the PLN is the network most capable of accurately predicting the high count projections it failed to generalise this to phantoms other than the one it was trained on. The results of the previous BSc Thesis study are displayed below in figure 9. The figure clearly shows the trained network was able to improve phantoms identical to those in the training set, but was unable to improve similar phantoms that it was not trained on. This would suggest the network was overfitted to the training set.

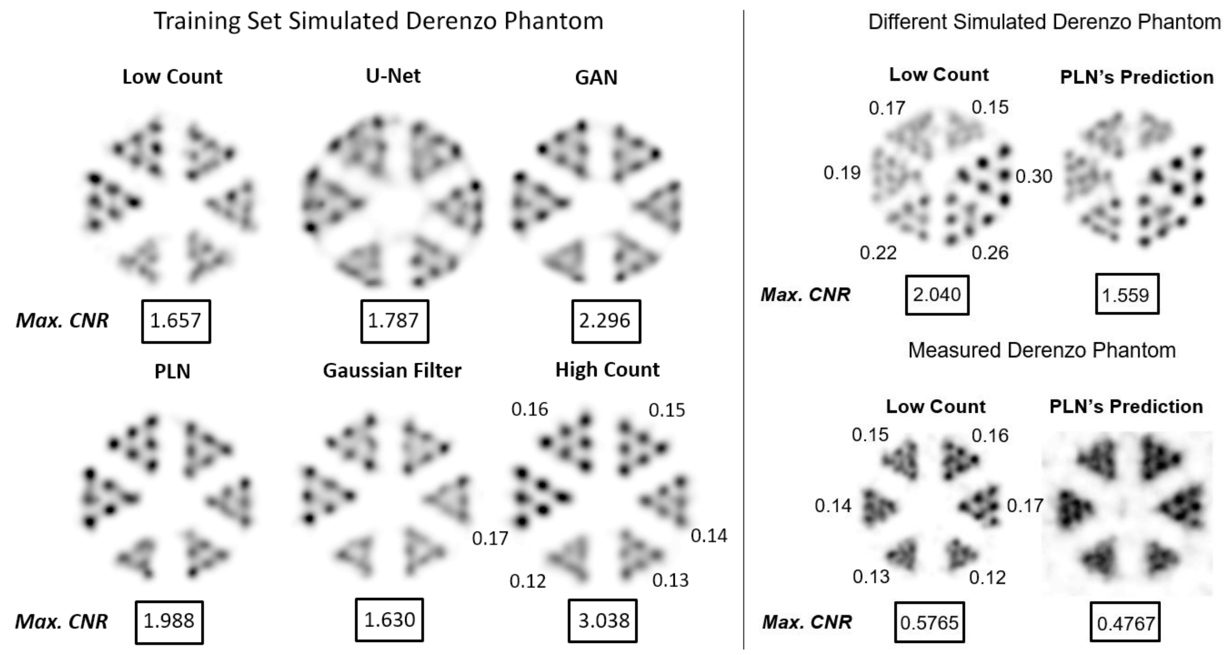


Figure 9: The results of the neural network from the previous BSc Thesis study. The reconstructions of a phantom identical to that inside the training set for different types of neural network as well as those for low and high count are displayed on the left together with their CNR. The reconstructions of two phantoms not inside the training set for the PLN are displayed on the right together with their CNR.

The idea is that by expanding the training dataset with projections of several diverse phantoms the PLN will be able to make its feature maps a bit more generalised, thus allowing the network to be more widely applicable. The phantoms contained within the expanded dataset can be found in table 1. For the exact details on each phantom simulated, please refer to the training set section in Appendix section A.

Table 1: A table displaying the phantoms inside the training dataset, together with their activity concentrations and the number of projections.

Phantom Type	Activity conc. (MBq/ml)	Background act. conc. (MBq/ml)	No. of projection pairs
Derenzo (6 segments)	4506	-	2358
Jaszczak	4506	-	1965
Uniform Cylinder	96	5	2586
Uniform Ellipsoid	243	-	2418
Derenzo (5 segments)	2463	2	2094

The expanded training set thus consists of 11.421 pairs of low and high count projections (so a total of 22.842 projection images), each of them at 512x512 resolution at full image size. In contrast the original training set used in the previous BSc Thesis study only contained 2.358 pairs of low and high count projections (a total of 4.716 projections) of 512x512 resolution at full image size, and these were all from the exact same phantom too. Now that the training set has been expanded to contain more and different phantom projections this should hopefully reduce the chance that the network will become overfitted.

As mentioned before all neural network code was written in Python 3 [10], executed using Jupyter Notebooks [11] and utilised the fast.ai library [12] to provide the neural network architectures. The original PLN constructed during previous BSc Thesis research trained only on the first phantom in table 1 will from here on be referred to as "PLN Original". PLN Original will be used to compare the newly created networks against. Three new neural networks were created and trained: PLN Expanded, PLN Expanded + Width Concatenation and PLN Expanded + Downsampling. These three networks are each detailed below in their own respective subsections.

3.1 PLN Expanded

The first new neural network that was made is PLN Expanded. The full code of PLN Expanded can be found in Appendix section C. PLN Expanded is very similar in architecture to the PLN Original with some minor tweaks here and there to file formatting and training parameters (such as batch size and learning rate). The main difference here of course is that PLN Expanded was trained on the fully expanded dataset detailed above. PLN Expanded is a neural network that uses a U-Net learner. The encoder for the U-Net was chosen to be a ResNet34 model pretrained on ImageNet [27] as it strikes a good balance between model depth/complexity and GPU memory requirement. The loss function was defined as Feature Loss using a VGG19 network with batch normalisation [28], also pretrained on ImageNet. The projections are loaded in as 512x512 grayscale images of 16-bit depth, which then get repeated twice to form a 3-channel image. For a detailed explanation of the image formatting please see section 5.1. PLN Expanded was run locally on an RTX 2070 SUPER with 8GB of GPU memory. Because of the large image resolution at full image size (512x512) the batch size was set to 8, as anything higher would require too much GPU memory. This is quite a small batch size, but this may actually be beneficial for training deep neural networks as suggested in [29]. The network is also put into half precision mode (FP16) during training as detailed in [30]. This means that values inside the neural network are stored as 16-bit wherever possible. The end result is that calculations can be sped up by 200% or more, and the available GPU memory is effectively doubled.

Like PLN Original, the neural network is initially frozen and first trained on projections downscaled to 256x256 at batch size 16 for 10 cycles, unfrozen, and trained for another 10 cycles. Frozen in this case means that the weights for all layers deeper than the final layer are static. Only the final fully-connected layer of the network is trained. This is called transfer learning and prevents the pretrained weights from changing. The pretrained initial layers extract only low-level features such as lines, curves, etc. and thus do not need training initially. Once unfrozen the weights of the deeper layers can also be adjusted. After this initial training the network is frozen again, then trained at full resolution at batch size 8 for 10 cycles, unfrozen, and trained for two further consecutive runs, each of 10 cycles. The learning rate starts of at 10^{-3} for the first 10 cycles and is then decreased gradually for the later cycles, a method called adaptive learning rate utilised in this case by the Adam optimiser [31]. This allows the network to take relatively large leaps towards the global minimum loss at first and then to steadily converge further. Once training is finished the network displays an input-prediction-target pair to show the general progress of the network after training (see figure 10).

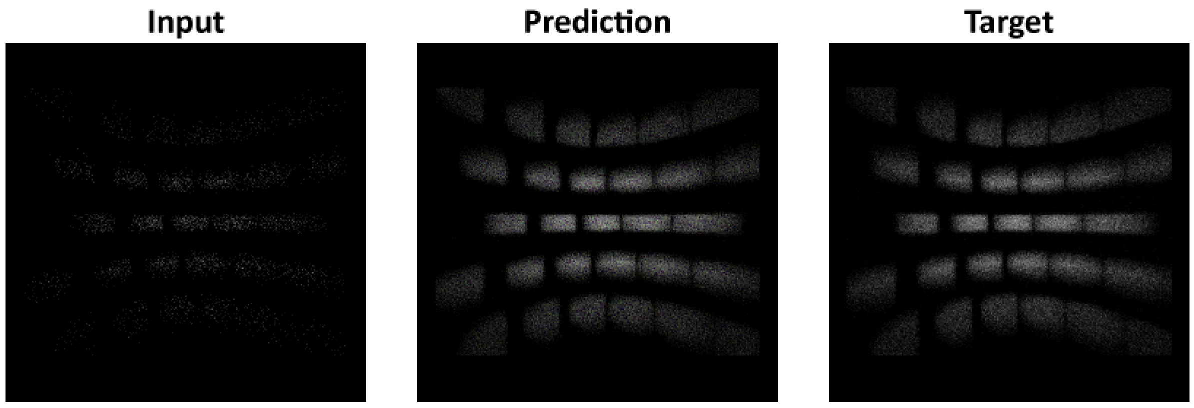


Figure 10: The progress of the neural network is displayed as an input-prediction-image pair after it is done training.

3.2 PLN Expanded + Concatenation

The second investigated method of improvement is concatenation. As is hopefully now clear PLN Expanded trains on the individual projections, which are 512x512 in size. These are split up by both detector plate (A, B, C) and by bed position (1, 2, ...). One of the strengths of deep learning is the ability for a neural network to pick up on details that are seemingly insignificant or not very noticeable to human beings and to use that information to produce a better prediction. Even though all these individual projections look similar to us in a general sense does not mean that they are all similar. After all, some projections were produced by detector plate A at bed position 3, and others were produced by detector plate C at bed position 9. At the same time of course the projections for detector plate B and C at bed position 3 were also produced, and likewise for the second example. The three projections created by detector plates A, B and C at the same bed position are related to one another. They are projections of the same object, in the same place, but "observed" if you will at different angles. Perhaps then there is some obscure information hidden inside those three matching projections that a neural network can extract useful information out of, but only if it can treat and train on all three projections together. This is not currently how PLN Expanded operates. PLN Expanded loads in all 22.842 projections, both low and high count, pairs them up and shuffles them randomly. The order in which the neural network sees the images is thus completely random, and it would not be able to extract any possible information from the matching detector plate projections. That is why two additional networks were constructed where the three matching detector plate projections are concatenated together into one image. This was done in two ways: through *width* concatenation and through *channel* concatenation.

3.2.1 Width Concatenation

The expanded dataset was pre-processed for both of these new networks with the help of some Python code. For the network that utilised *width* concatenation the three individual matching projections of size 512x512 were put side by side and stitched together into one big, rectangular image of size 1536x512 (width x height). The projections produced by detector plate A were placed in width position 1 through 512, those produced by detector plate B in width position 513 through 1024 and those produced by C in width position 1025 through 1536. An example low count high count pair with this type of concatenation is displayed in figure 11.

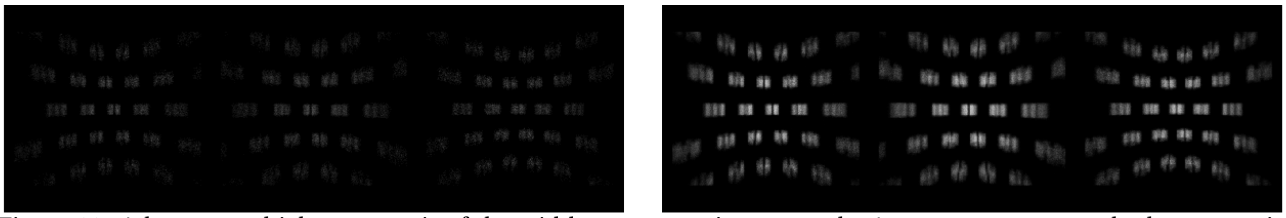


Figure 11: A low count high count pair of the width concatenation network. As one may assume the low count is displayed on the left and the high count is displayed on the right.

These large concatenated images are still grayscale, so in order to train on them they must be replicated twice to form a 3-channel 1536x512 image of 16-bit depth. This width concatenated dataset still takes up the exact same size on disk. This is because although the images now each have three times as many pixels, the number of total projections has been reduced down to a third. This dataset therefore has 7614 projections in it, or 3807 pairs. The network however is going to need some adjustments in order to be able to deal with these images. This is because although the total size on disk of these images is the same as the unconcatenated dataset, the input to the network is now three times as large. This means that each subsequent layer in the neural network is also going to have to be wider. It is in this case not the training dataset that is a strain on the GPU memory, but the network itself.

The RTX 2070 SUPER with its 8GB of GPU memory is not good enough to be able to run this network, and so the aforementioned cloud-based GPU rental service Paperspace [13] will be used. The service allows you to rent a strong GPU for free for a limited amount time. In this case an NVIDIA Quadro P5000 with 16GB of GPU memory was rented for 6 hours at a time, which is the maximum allowed time for non-paying users by the service. The width concatenated dataset and Jupyter notebook were uploaded to the cloud storage also provided by Paperspace and run for as long as the 6 hour time window allowed. The network was first trained on downsized projections of 768x256 at batch size 4 for 10 cycles, unfrozen, and trained for another 10 cycles. The resolution was then upped to the full 1536x512, the batch size was reduced down to 2 and the network was frozen again. The network was trained for 6 cycles, unfrozen again, and trained again for two consecutive runs of 5 cycles each. This is less cycles than the regular PLN Expanded was trained for in total, but the increased neural network size also brings with it additional required computing time. As the GPU was only available for 6 hours at a time, 6 or 5 cycles was the maximum number that was possible in that time window.

3.2.2 Channel Concatenation

For the network that utilised *channel* concatenation the three matching projections were put into their own separate channel to create a 3x512x512 image. The projections produced by detector plate A were placed on the first channel, those produced by detector plate B on the second channel and those produced by C on the third channel. An example low count high count pair with this type of concatenation is displayed in figure 12. As can clearly be seen from the figure the fast.ai library has chosen to display the image as if it were an RGB image. This is not representative of the actual colours of this image as it is in essence just a three high stack of 1-channel images, but is useful for showing these separate channels in a way that still makes them distinguishable from one another.

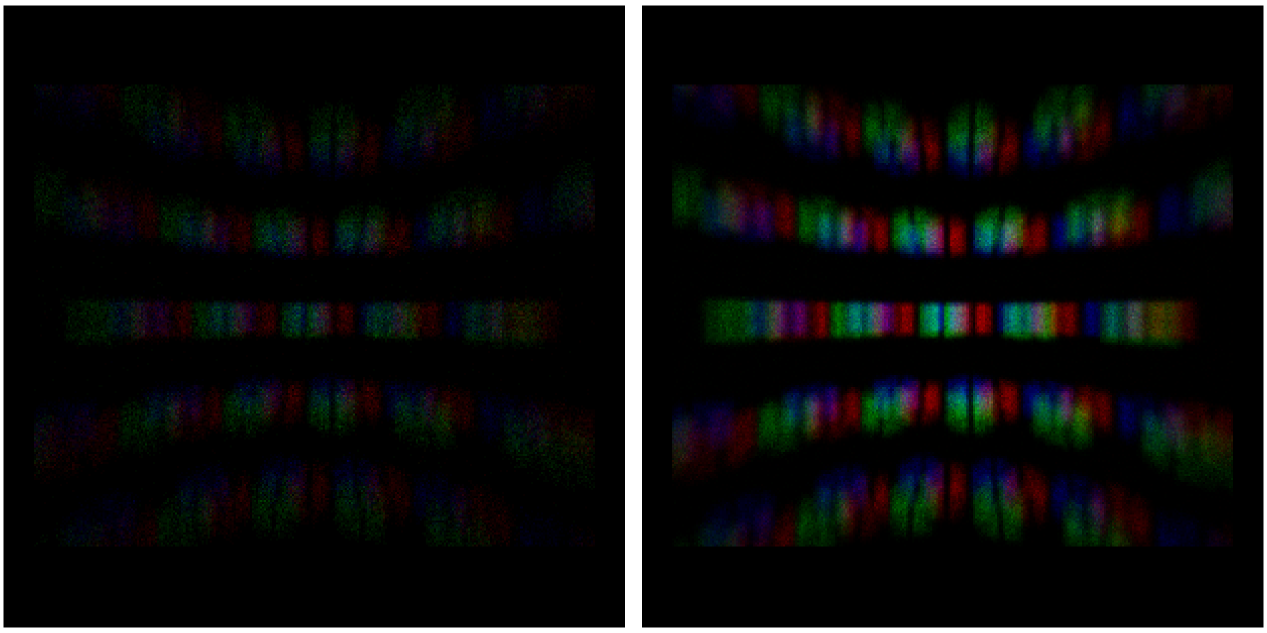


Figure 12: A low count high count pair of the channel concatenation network. The low count (left) and high count (right) clearly display how the three detector projections are placed on each colour channel.

As the input and output to the neural network are identical between this dataset of channel concatenated images and the other dataset no adjustments to the PLN Expanded were needed. A copy of the network was made and trained (locally) on this concatenated dataset using the exact same parameters. Once the network had been fully trained for a total of 50 cycles a test was performed to see if there were any issues with this approach to combining the three detector plates. Unfortunately it was then discovered that training the network with the three projections placed on different channels caused the network to leave imprints of each layer on each other layer. When these projections are then pried back apart it leaves a lot of pixel intensity in the image where there is not supposed to be any. Because of this it was decided that rather than continuing with this *channel* concatenation setup it would be best to only have *width* concatenation instead.

3.3 PLN Expanded + Downsampling

A third and final method of improvement was explored: downsampling. Or rather multisampling directly followed by downsampling. This method of improvement was a late addition to the study and heavily inspired by a similar method applied in the video games industry. Supersampling Anti-Aliasing (or SSAA for short) is a technique to counteract aliasing (pixelated edges). The technique works by rendering the image at a higher resolution than the desired final output and then downsampling it again to obtain an image with smoother edges than the regular image would have had. SSAA is very much a brute-force method of obtaining an image with smoother edges as the entire image is rendered at a higher resolution, requiring much more GPU memory as well as memory bandwidth. Because of this MSAA (Multisampling Anti-Aliasing) is favoured nowadays over SSAA as it only supersamples around edges. However SSAA will do just fine for this study as supersampling only needs to be performed on a small number of images. The process of downsampling is detailed below in figure 13.

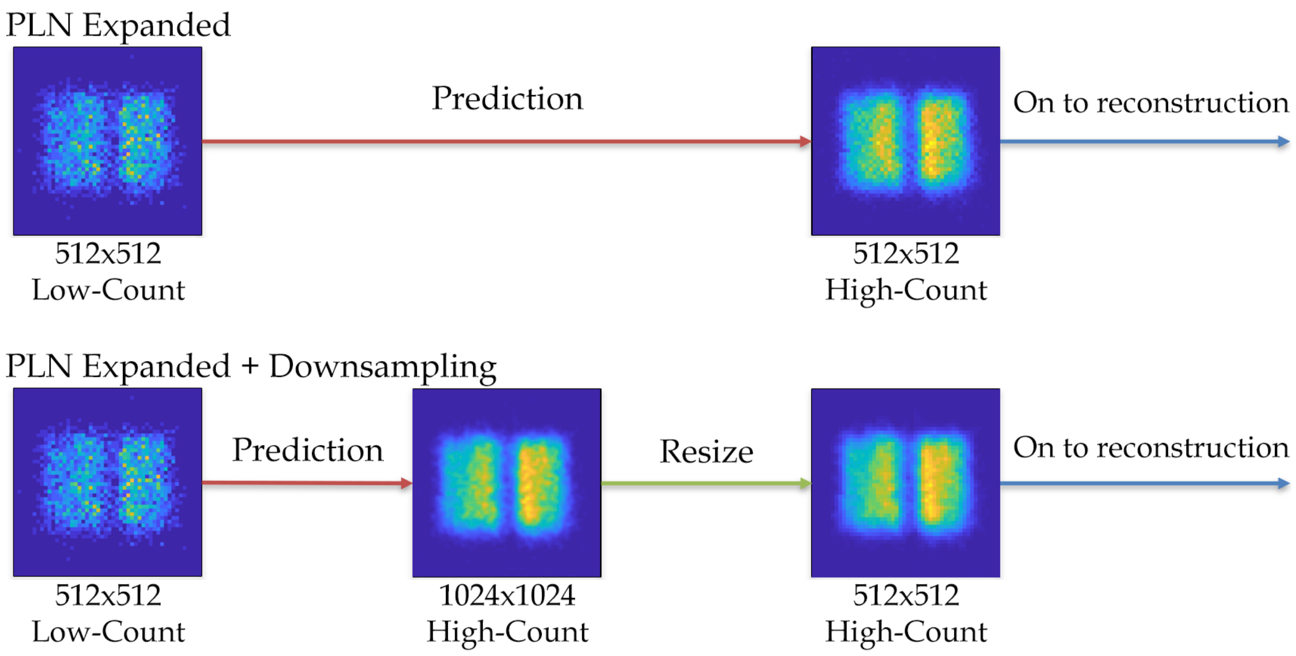


Figure 13: The prediction process as it is performed by PLN Expanded normally (top) and as it is performed using downsampling (bottom).

The 512x512 low count projections of the test sets are normally directly predicted to 512x512 high count projections. Now however, the 512x512 low count projections will first be improved to 1024x1024 high count projections, which will then be resized back down to size 512x512. The end result is a predicted high count projection that has much smoother edges, as well as seemingly reduced noise. The trained PLN Expanded network was used for producing predictions directly. No new network was trained to perform this task.

3.4 Evaluation

The performance of the neural networks is evaluated with the use of seven test sets. These test sets were chosen to have varying structures, sizes and levels of activity in order to determine which neural networks perform better on different kinds of projections. The exact details for each test set can be found in Appendix section B.

The first four test sets used were of Derenzo-like phantoms.

Simulated Derenzo phantom 1 is identical to the projections inside both the original and the expanded training set. The high count projections (100%) were simulated at 3600 MBq/ml, the low count projections (10%) were simulated at 360 MBq/ml, and the phantom was simulated for 3 hours. The low count projections were used for improvement by the neural networks.

Simulated Derenzo phantom 2 is not in either of the training sets. The high count projections (100%) were simulated at 212 MBq/ml, the low count projections (10%) were simulated at 21.2 MBq/ml, and the phantom was simulated for 3 hours. The low count projections were used for improvement by the neural networks.

Simulated Derenzo phantom 3 is identical to simulated Derenzo phantom 2, but the high count (100%) projections were used for improvement by the neural networks. This test set is also not in either of the training sets.

The measured Derenzo phantom was, as the name suggests, physically measured as opposed to simulated and is not in either of the training sets. The high count projections (100%) were measured at 4510 MBq/ml, the low count projections (10%) were measured at 451 MBq/ml, and the phantom was measured for 3 hours. The low count projections were used for improvement by the neural networks.

Three uniform cylinder phantoms were also added as test sets.

Simulated uniform cylinder 1 is not in either of the training sets. Specific activity concentrations were not known for this phantom. The high count projections are at 100% count, the low count projections are at 10% counts. The low count projections were used for improvement by the neural networks.

Simulated uniform cylinder 2 is identical to simulated uniform cylinder 1, but the high count (100%) projections were used for improvement by the neural networks. This test set is therefore also not in either of the training sets.

A measured uniform cylinder was added as a test set too. The measured uniform cylinder is not in either of the training sets. The high count projections (100%) were measured at 38.6 MBq/ml, the low count projections (10%) were measured at 3.86 MBq/ml, and the phantom was measured for 3 hours. The low count projections were used for improvement by the neural networks.

The quality of the reconstructions will be examined both quantitatively as well as qualitatively. The results of the Derenzo-type phantoms will be made quantitative using the Contrast-to-Noise Ratio (CNR), defined as it was in [32, 15]. The CNR of each rod sector is defined as C_s/N_s. C_s is the contrast of each rod sector and is defined as

$$C_s = \frac{\overline{I_s} - \overline{B_s}}{\overline{I_s}}, \tag{2}$$

where $\overline{I_s}$ is the mean intensity over the activity regions of sector s, and $\overline{B_s}$ is the mean intensity over the background regions of sector s. N_s is the noise of each rod sector and is defined as

$$N_s = \frac{\sqrt{{\sigma_{I_{s,p}}}^2 + {\sigma_{B_{s,p}}}^2}}{\overline{IB_s}}, \tag{3}$$

where ${\sigma_{I_{s,p}}}^2$ and ${\sigma_{B_{s,p}}}^2$ are the standard deviations over I_s and B_s respectively calculated over all sectors s and all planes p, and where IB_s is the mean intensity over all regions of interest (ROIs) in sector s. All of these calculations have been automated with a MATLAB script written by M.P. Nguyen, so that the CNR for various phantoms can now easily be plotted.

A low count reconstruction of simulated Derenzo phantom 1 was passed through the script and the CNR was calculated for each iteration, for each rod sector. The results have been plotted in figure 14 below.

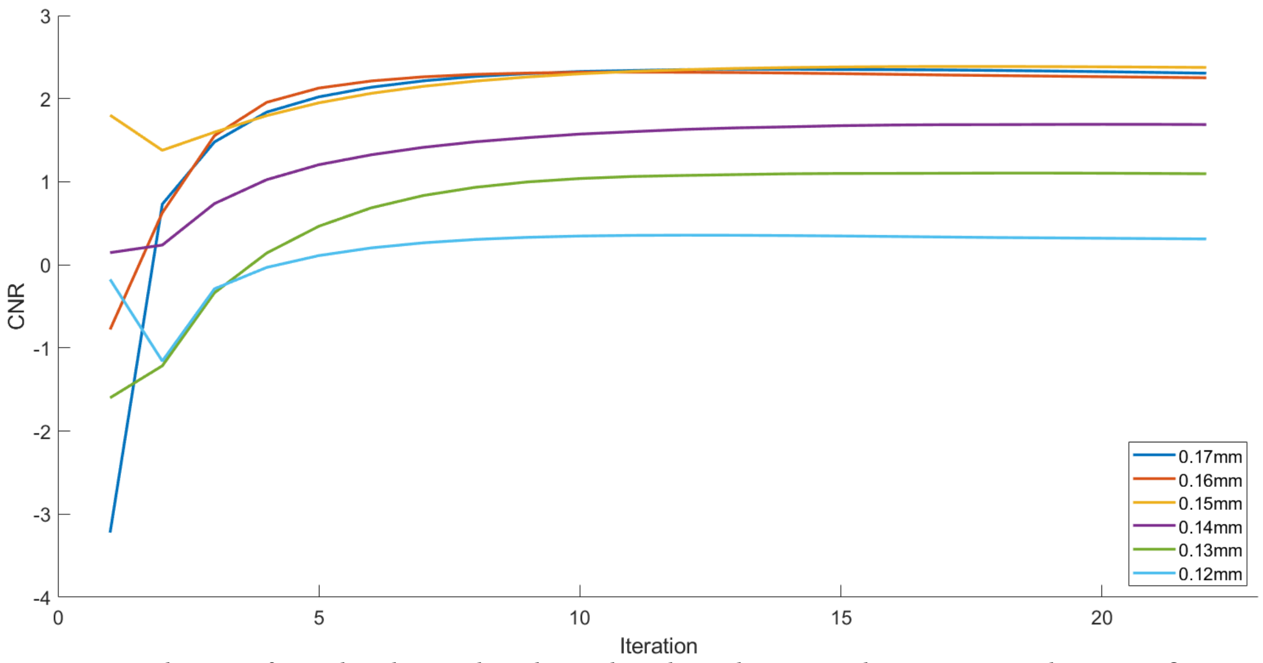


Figure 14: The CNRs for each rod sector have been plotted together versus the iteration number in one figure.

This graph alone clearly shows that the rod sectors of 0.17mm, 0.16mm and 0.15mm are going to be of comparable quality, while the rod sectors of the smaller rods are going to be progressively worse. Ideally of course a plot like this would be made for each test set, each reconstruction type and each rod size. As that would result in 20 figures though the results have been condensed down somewhat. In the sections below you will find tables detailing the maximum CNR value for each combination of reconstruction type and rod size, as well as figures with a more qualitative evaluation showing the best overall reconstructions together with their CNR. It is worth noting that the values displayed in the tables are the maximum CNRs for each rod sector size across all iterations. The CNR values do therefore not necessarily belong to the same iteration.

The quality of the uniform phantoms was made quantitative by calculating the uniformity (sometimes known as the coefficient of variation) of each reconstruction. The uniformity is a measure calculated from the standard deviation and mean across (in this case) 5 ROIs in the reconstruction [32]. Specifically, the equation to calculate the uniformity is

$$\text{Uniformity} = 100 \cdot \frac{\sigma_u}{\overline{u}}, \tag{4}$$

where $\overline{u}$ represents the average value from the 5 ROIs, and σ_u is the standard deviation between the 5 ROI means. Uniformity is expressed as a percentage, where lower values mean more uniform phantoms.

4 Results

4.1 Simulated Derenzo Phantom 1

Simulated Derenzo phantom 1 is identical to the Derenzo phantom projections contained within both the original and the expanded training set. It is therefore expected that all neural networks should be able to produce CNR values higher than those of the low count reconstruction. Table 2 displays the quantitative results of each reconstruction for each rod sector. As was expected the neural networks outperform the low count reconstruction on all rod sectors. PLN Expanded + Downsampling is the clear victor here, producing a CNR that is on average 87.0% better than the low count reconstruction. The other three networks produced similar results to one another: PLN Expanded + Width Concatenation achieved 59.7% better CNR on average, PLN Expanded achieved 53.1% better CNR on average and PLN Original achieved 54.5% better CNR on average; all significantly lower than PLN Expanded + Downsampling.

Table 2: The CNR for each reconstruction type for each rod sector size of simulated Derenzo phantom 1.

Reconstruction	**0.17mm**	**0.16mm**	**0.15mm**	**0.14mm**	**0.13mm**	**0.12mm**
Low Count	2.355	2.319	2.387	1.691	1.104	0.357
High Count	5.379	5.595	3.612	3.156	1.391	0.422
PLN Original	2.633	3.441	3.279	2.182	2.050	0.767
PLN Expanded	2.825	3.912	3.476	2.265	1.537	0.754
PLN Expanded + Width Concat.	2.944	3.955	3.328	2.161	1.892	0.800
PLN Expanded + Downsampling	3.152	3.762	4.506	2.611	2.349	0.963

Moving on the the more qualitative analysis of the reconstructions figure 15 shows that the PLN Expanded + Downsampling manages to produce the highest CNR here too. Out of the four neural network reconstructions PLN Original seems to have performed the worst as can be concluded by both the average CNR and visual examination, since artifacts can be seen sprouting out of the what are supposed to be circular dots. The three networks trained on the expanded dataset all seem similar in quality for the larger rod sectors, but if only considering the smaller rod sectors it becomes clear that PLN Expanded + Downsampling is the nicer reconstruction.

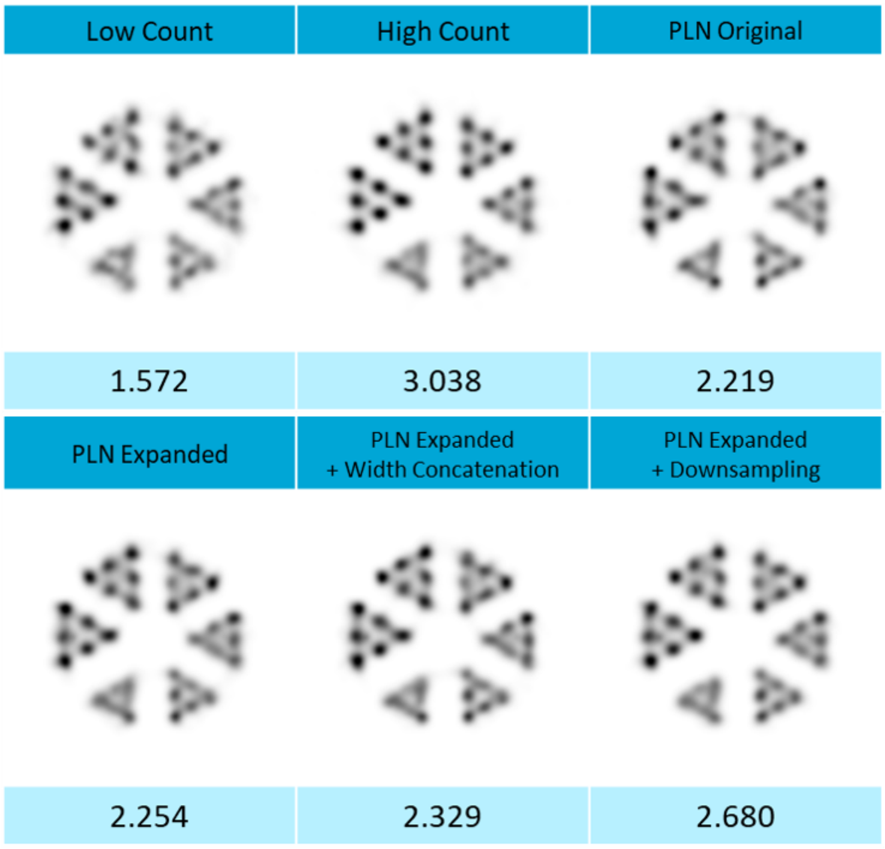


Figure 15: The best reconstructions with the highest CNR for each neural network are lined up besides one another together with the low and high count reconstruction for comparison. The average CNR value for each reconstruction is displayed below it.

4.2 Simulated Derenzo Phantom 2

The projections of simulated Derenzo phantom 2 were not inside the training set, making it well-suited as a test to see how generalised the networks are. Table 3 displays the quantitative results of each reconstruction for each rod sector. Against expectation the neural networks that were trained on the expanded dataset performed significantly worse than PLN Original and even the low count reconstruction in some cases. Possible suggestions as to why this is the case will be laid out in section 5. PLN Original produced a reconstruction somewhat better than the low count at a 30.1% higher average CNR. PLN Expanded, PLN Expanded + Width Concatenation and PLN Expanded + Downsampling all performed worse than the low count at a 15.2%, 10.8% and 0.6% lower average CNR across the best CNR values of all iterations.

Table 3: The CNR for each reconstruction type for each rod size of simulated Derenzo phantom 2.

Reconstruction	**0.30mm**	**0.26mm**	**0.22mm**	**0.19mm**	**0.17mm**	**0.15mm**
Low Count	6.762	3.600	2.272	1.149	0.254	0.299
High Count	11.713	6.775	8.167	3.039	1.911	0.735
PLN Original	7.361	3.753	2.954	1.306	0.621	0.238
PLN Expanded	4.279	2.626	1.723	0.874	0.318	0.286
PLN Expanded + Width Concat.	3.540	2.570	2.000	0.959	0.339	0.318
PLN Expanded + Downsampling	5.046	2.899	1.969	1.091	0.447	0.250

Looking at the reconstructions in figure 16 the extent of the decrease in quality can be seen. The networks trained on the expanded dataset performed much worse than the PLN Original. The PLN Original was only able to achieve a marginally better average CNR, and even then one could argue that the low count reconstruction is qualitatively better than the one produced by PLN Original.

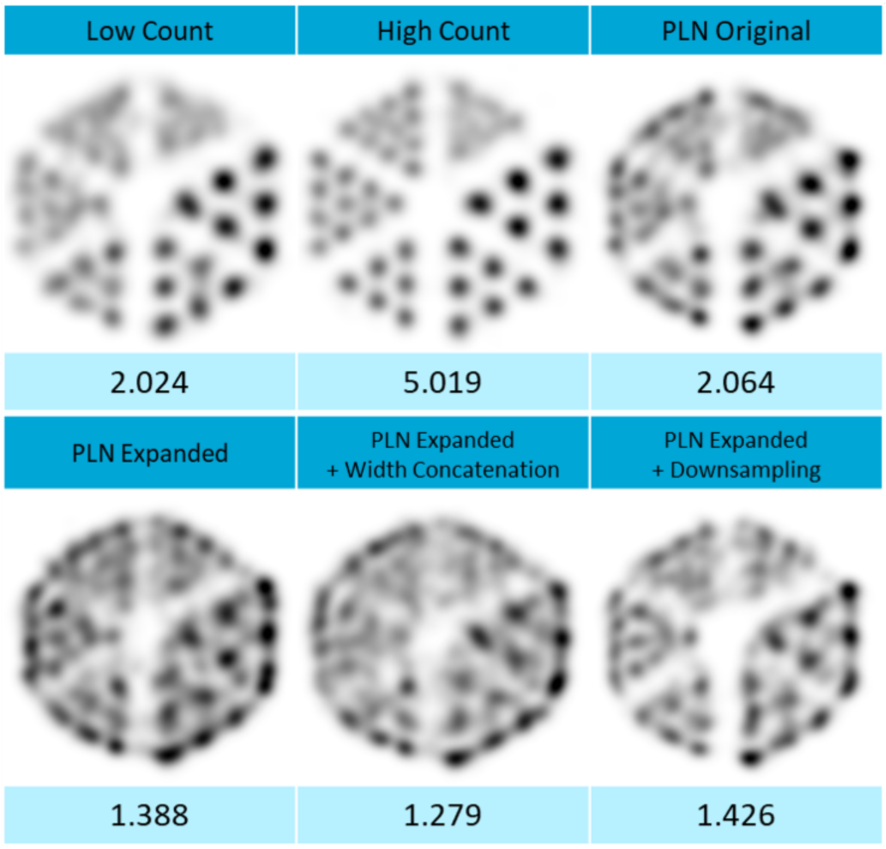


Figure 16: The best reconstructions with the highest CNR for each neural network are lined up besides one another together with the low and high count reconstruction for comparison. The average CNR value for each reconstruction is displayed below it.

4.3 Simulated Derenzo Phantom 3

Simulated Derenzo phantom 3 is identical to simulated Derenzo phantom 2, but the high count projections of simulated Derenzo phantom 2 were taken as the "low count" input of the neural network. As the projections are already at 100% counts there is no "high count" reconstruction available for this test set. Just like simulated Derenzo phantom 2, this test set was not inside any of the training sets. Table 4 displays the quantitative results of each reconstruction for each rod sector. The low count reconstruction has a high CNR per rod sector size. This time around the neural networks performed much better and were able to improve the reconstruction by quite a bit. PLN Original produced a reconstruction with 9.8% higher average CNR, PLN Expanded one with 18.1% higher average CNR, PLN Expanded + Width Concatenation one with 21.6% higher average CNR and PLN Expanded + Downsampling produced a reconstruction with 32.8% higher average CNR.

Table 4: The CNR for each reconstruction type for each rod size of simulated Derenzo phantom 3.

Reconstruction	**0.30mm**	**0.26mm**	**0.22mm**	**0.19mm**	**0.17mm**	**0.15mm**
Low Count	11.713	6.775	8.167	3.039	1.911	0.735
PLN Original	12.482	9.604	5.834	3.150	2.317	0.841
PLN Expanded	11.968	11.679	6.297	3.593	2.297	0.872
PLN Expanded + Width Concat.	12.071	10.248	6.615	4.033	2.057	1.131
PLN Expanded + Downsampling	12.958	11.047	7.843	3.878	2.866	1.101

Looking at figure 17 the reconstructions appear to be quite nice this time. Some of the smaller rod sectors can even be made out this time. PLN Original is showing the most artifacts out of all four neural network reconstructions. Visually the other three neural network reconstructions look very similar, so resorting to the average CNR value for each reconstruction tells us that PLN Expanded + Downsampling has produced the best reconstruction here.

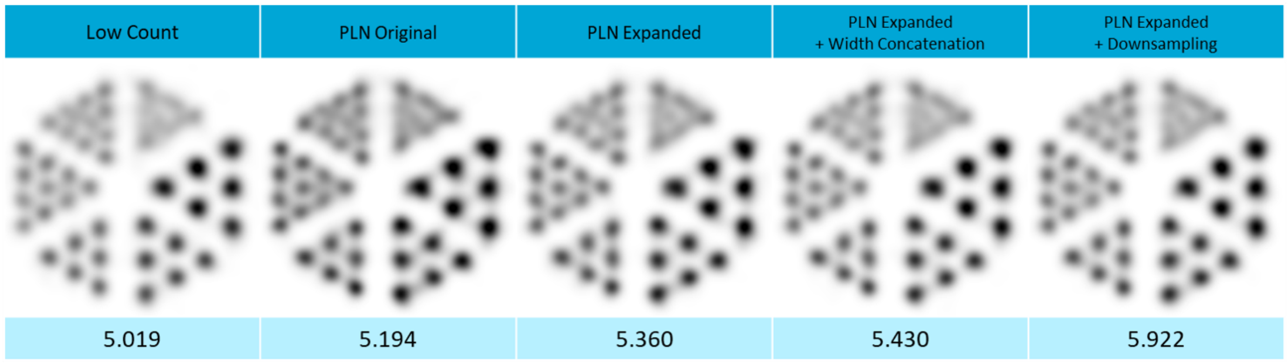

Low Count
PLN Original
PLN Expanded
PLN Expanded + Width Concatenation
PLN Expanded + Downsampling
5.019
5.194
5.360
5.430
5.922

Figure 17: The best reconstructions with the highest CNR for each neural network are lined up besides one another together with the low count reconstruction for comparison. The average CNR value for each reconstruction is displayed below it.

4.4 Measured Derenzo Phantom

The measured Derenzo phantom is similar to the simulated Derenzo phantom 1 in that it has rod sector sizes ranging from 0.17mm to 0.12mm. It is however mirrored and has some imperfections due to the fact that it was measured. The projections from this test set were not in any of the training sets. Table 5 displays the quantitative results of each reconstruction for each rod sector. The measured phantom is difficult to improve for all four networks as it seems, but PLN Expanded + Downsampling seems to be performing the best based on this table. Compared to the low count reconstruction PLN Original has an average CNR that is 11.2% lower. PLN Expanded has an average CNR that is 3.0% higher. PLN Expanded + Width Concatenation performs a little better at 25.8% higher average CNR. The largest improvement comes from PLN Expanded + Downsampling. It achieved an 85.3% higher average CNR.

Table 5: The CNR for each reconstruction type for each rod size of the measured Derenzo phantom.

Reconstruction	**0.17mm**	**0.16mm**	**0.15mm**	**0.14mm**	**0.13mm**	**0.12mm**
Low Count	1.875	1.369	0.563	0.433	0.115	0.373
High Count	2.890	2.814	1.003	1.134	1.033	1.013
PLN Original	2.142	0.853	0.155	0.041	0.245	0.396
PLN Expanded	1.865	1.195	0.302	-0.117	0.345	0.391
PLN Expanded + Width Concat.	2.419	1.439	0.587	0.183	0.310	0.390
PLN Expanded + Downsampling	2.559	1.598	1.199	0.431	0.513	0.373

The reconstructions in figure 18 too seem to indicate that PLN Expanded + Downsampling produced by far the best reconstructions out of all four neural networks. Interestingly enough it also appears to have significantly reduced background activity surrounding the phantom when compared to the other neural networks.

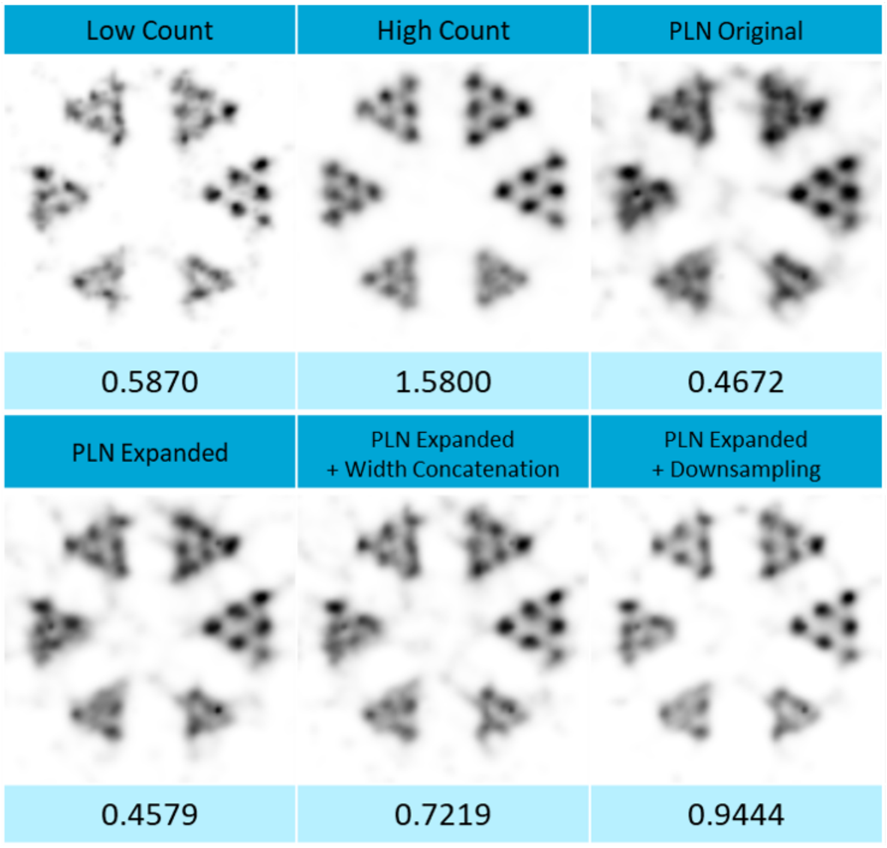
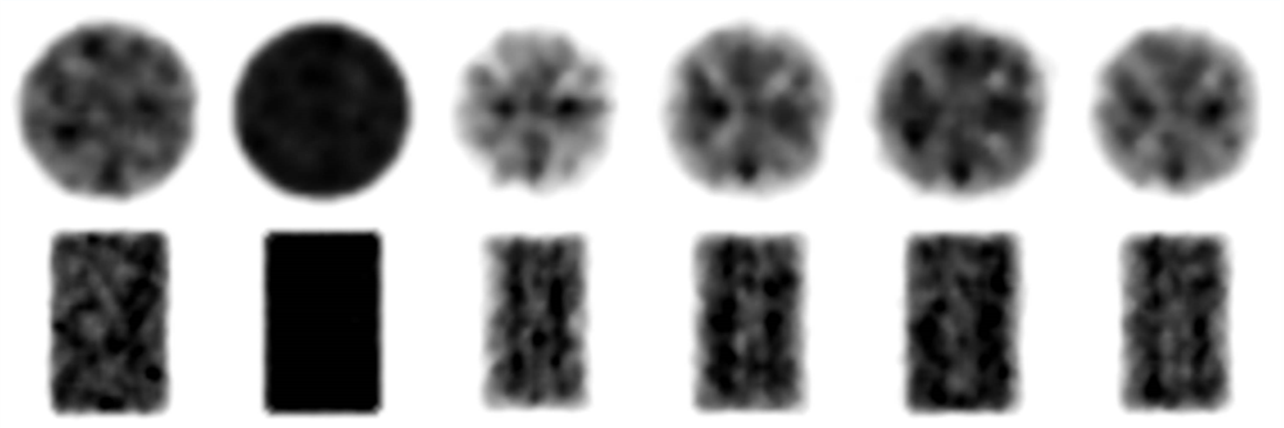


Figure 18: The best reconstructions with the highest CNR for each neural network are lined up besides one another together with the low and high count reconstruction for comparison. The average CNR value for each reconstruction is displayed below it.

4.5 Simulated Uniform Cylinder 1

The expanded training set does contain uniform cylinder projections, and so having some test sets that are of uniform cylinders seemed interesting. The projections from this test set specifically were not in any of the training sets. This uniform cylinder phantom was simulated with a length of 1.5mm and 6mm in diameter. The axial and coronal cross section for each reconstruction have been displayed in figure 19 together with the corresponding uniformity. All neural networks seem to decrease the uniformity when compared to the low count reconstruction (recall that lower values are better). PLN Expanded + Downsampling appears to be the best performing neural network in this case, reaching a uniformity of 1.929%. Remarkably, PLN Original is quite close behind with 2.147% while PLN Expanded and PLN Expanded + Width Concatenation only achieve marginal improvement over the low count uniformity of 3.171%. This result appears to be somewhat in line with what was observed in subsection 4.2, where the neural networks trained on the expanded training set largely failed to improve on the low count phantom because of the very low count nature of the projections.

Low Count	High Count	PLN Original	PLN Expanded	PLN Expanded + Width Concatenation	PLN Expanded + Downsampling



3.171%	0.843%	2.147%	2.821%	2.707%	1.929%

Figure 19: An axial cross section (top) and coronal cross section (bottom) are lined up besides one another together with the low and high count reconstruction for comparison. The uniformity value for each reconstruction is displayed below it.

4.6 Simulated Uniform Cylinder 2

A second uniform cylinder test set was simulated to see how the neural networks would handle more high count projections of uniform cylinders. Simulated uniform cylinder 2 is identical to simulated uniform cylinder 1, but the high count projections of simulated uniform cylinder 1 were taken as the "low count" input of the neural network. As the projections are already at 100% counts there is no "high count" reconstruction available for this test set. The projections from this test set were not in any of the training sets. The axial and coronal cross section for each reconstruction have been displayed in figure 20 together with the corresponding uniformity. Looking at the figure it seems the neural networks have all made the uniformity of the reconstructed phantoms worse. This is visible qualitatively in the axial and coronal cross sections as well. The neural network reconstructions all seem to be less dense in the centre of the cylinders compared to the edges. This is an interesting result. The three PLN Expanded networks have all been trained on uniform cylinder projections, so the fact that these networks seem to lead to uniformities even worse than that of PLN Original is somewhat against expectations.

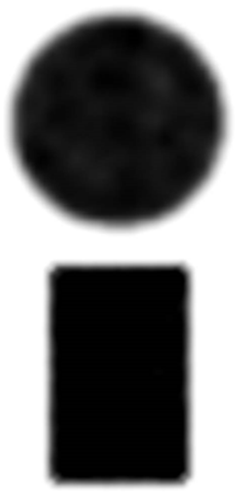
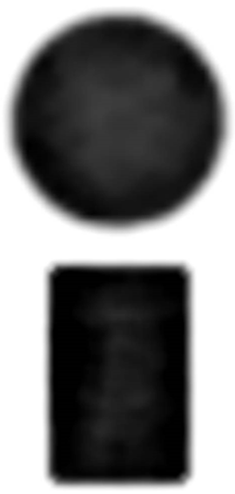
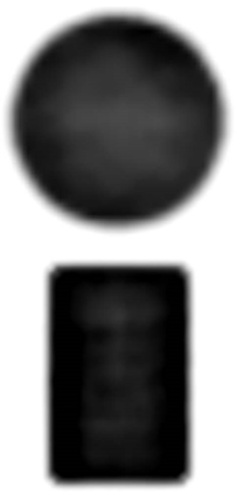
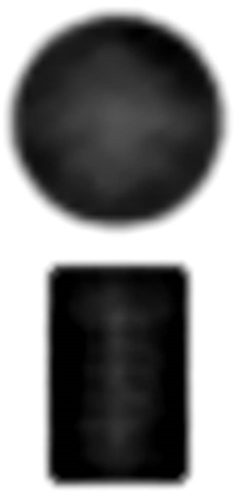
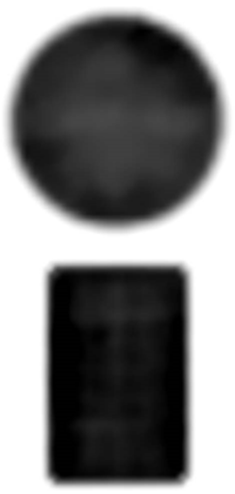
Low Count	PLN Original	PLN Expanded	PLN Expanded + Width Concatenation	PLN Expanded + Downsampling
				
0.843%	1.204%	2.294%	2.072%	3.093%

Figure 20: An axial cross section (top) and coronal cross section (bottom) are lined up besides one another together with the low count reconstruction for comparison. The uniformity value for each reconstruction is displayed below it.

4.7 Measured Uniform Cylinder

A measured uniform cylinder test set was also made to see if the results achieved on the simulated uniform cylinders can be applied to measured uniform cylinder as well. The projections from this test set were not in any of the training sets. The axial and coronal cross section for each reconstruction have been displayed in figure 21 together with the corresponding uniformity. Looking at the uniformity values it seems the neural networks were all unable to improve the projections properly. The uniformities are all much higher than that of the low count. By far the worst neural network reconstruction is that of PLN Expanded at a uniformity of 8.267%. This uniformity is 5x higher than the low count uniformity of 1.652%. The likely reason for this massive decrease in quality can be seen visually in the figure. All neural network reconstructions contain artifacts in the centres of the phantom. These artifacts, areas of lower activity density, are especially present in the reconstructions of PLN Expanded and PLN Expanded + Width Concatenation. The other two neural networks, PLN Original and PLN Expanded + Downsampling, also show these artifacts but they are not quite as obvious, and this is in agreement with their lower uniformity values of 4.688% and 4.031% respectively. Though even these uniformities are still much worse than the normal low count reconstruction.

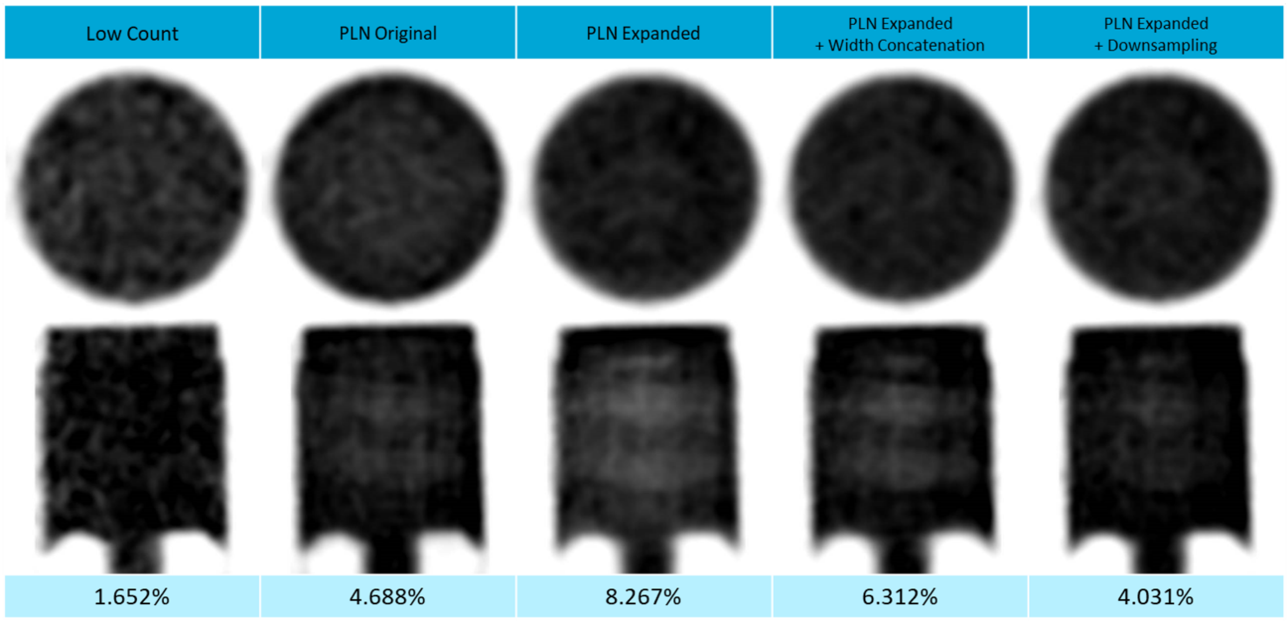


Figure 21: An axial cross section (top) and coronal cross section (bottom) are lined up besides one another together with the low count reconstruction for comparison. The uniformity value for each reconstruction is displayed below it.

5 Discussion

After examining the reconstructions produced for the various phantoms by the neural networks it became clear that expanding the dataset on which the neural networks are trained on is beneficial to some projections, but not nearly to all projections. One constraint for the neural networks seems to be the level of activity at which the projections are made.Very low activity projections like simulated Derenzo phantom 2 and simulated uniform cylinder 1 seem to be difficult to improve for the networks. Out of all four networks only the PLN Original seemed able to improve simulated Derenzo phantom 2. One possible explanation for this could be that because of the low activity (and thus low counts) the neural networks are having difficulties figuring out how to improve these projections. Especially the three neural networks that were trained on the expanded training set seem to be having issues. This would fit with the proposed reason for the decrease in quality. The three networks trained on the expanded dataset have seen many more structures than the PLN Original. This might make it more difficult for those networks to figure out whether the projection at hand is a Derenzo-like phantom or perhaps a more uniform phantom. Meanwhile the PLN Original was only ever trained on Derenzo-like phantom projections, and would thus have no reason to improve the projections as if they were anything else. A future improvement might therefore be to expand the dataset even further with very low count projections.

This leads to an important point on neural networks. Neural networks like these are always trained to take some input and transform it into a desired target output. Once training is done and it is time for the network to produce prediction images with no desired target present, there can be no guarantee that what the neural networks produce is actually going to be representative of the ground truth. People should therefore always keep this fact in mind when working with projections and reconstructions produced by neural networks. Just because a neural network can greatly improve a projection in one particular situation does not mean it will always be able to do this, especially when applied to situations the network is unfamiliar with.

The PLN Expanded network does not benefit much from the additional training data when it comes to Derenzo-like phantoms it seems, the CNR only being 0.1-0.2 higher than the PLN Original at best. Training on the projections when they are width concatenated seems to improve the neural network's ability to predict Derenzo-like phantoms very slightly, but again the CNR is only increased by at most 0.3 when compared to the PLN Original. The largest improvement came from the late addition to the study: downsampling. PLN Expanded + Downsampling was surprisingly effective at improving Derenzo-like phantoms. Its CNR when compared to the PLN Original was consistently much higher (leaving simulated Derenzo phantom 2 out of the comparison). The largest increase was seen for simulated Derenzo Phantom 3, which went from a CNR of 5.194 from the PLN Original to a CNR of 5.922 with the downsampling network. The measured Derenzo phantom too saw a large increase in CNR from the downsampling network. Now that this neural network seems able to improve Derenzo phantoms reasonably well, it would be interesting to in the future include scatter in the simulations to see whether the network can be applied effectively there as well.

As the PLN Expanded + Downsampling was a late addition to the study it was not properly explored as well as it could have. The already trained model from the regular PLN Expanded network was used to create the predictions in 1024x1024, which were then downsampled back to 512x512. The only tweak as it were was made in the post-processing of the projections during the prediction process, not in the neural network itself. It is therefore recommended to construct a neural network specifically designed and trained to increase the projection quality from low to high count and at the same time increase the quality from low to high resolution.

One way this could be done is to create a separate dataset with only the low count projections, but downsized to for example 96x96. The first few cycles of training could then be focused on for example improving these low count 96x96 projections scaled up to 128x128 towards the target projections, which would be the high count projections scaled down to 128x128. Once the neural network has been trained sufficiently one could move on to a larger resolution. The 96x96 low count projections are scaled up to 256x256 and trained towards the target, which would then be the 512x512 high count projections scaled down to 256x256. Moving on to the final step, the low count 96x96 projections would be scaled up to 512x512 and trained towards the target, which would be the high count 512x512 projections as is. This way the neural network should learn to both increase image quality in the sense of low to high resolution as well as image quality in the sense of low to high count.

Another improvement to this could be using PLN Expanded + Width Concatenation as a base rather than the regular PLN Expanded, as it showed slight improvement in CNR. Finally, the upscaling in this study was performed from 512x512 to 1024x1024. Perhaps it would be interesting to see whether a larger upscaling step would be more or less beneficial, say 2048x2048. Ideally these recommendations would be combined and applied together. However the resource requirements for a network that applies both width concatenation and upscales to 1024x1024 or 2048x2048 would be considerable. It is therefore up to future research to find out whether this is feasible or not.

5.1 A note on image formatting

Upon closer examination of the PLN Original code there appeared to be an issue with the way that the projection files were being loaded in for training. All projection files are grayscale 512x512 images with a 16-bit depth stored in Tiff format. Ideally you would then like a neural network that loads these Tiff files in that exact way and trains on them as such, however this was not currently the case. The Tiff files were opened and loaded using default fast.ai functions, which interpreted the images as 3-channel images of 8-bit depth. The Python module Pillow [33] was initially used to load the images in as 1-channel images of 16-bit depth. Both the U-Net learner and the Feature loss network needed manual adjustments as well to allow for an input and output that was not 3-channel but 1-channel. After training and reconstruction it was concluded that the quality of the reconstruction produced was quite a lot worse than the ones produced by PLN Original. The projections also seemed to be of lower quality and contained checkerboard artifacts. It was suggested online by people having similar issues with grayscale training that this may be because the networks were pretrained on RGB images (ImageNet) and thus are unfit for training on grayscale images, though there is no clear consensus or definitive explanation for why this happens.

3-channel images with 16-bit depth seemed required for the neural network to produce proper predictions. As Pillow does not currently support multi-channel images of 16-bit depth, the Python module Tifffile [34] was installed and used to properly load in the projections. Custom functions and classes were written and adapted from the fast.ai library to allow the use of this new module. This however created yet another problem. Fast.ai networks require the pixel values to be normalised to lie between 0 to 1. For a regular 8-bit image this means that every pixel value is divided by 255, the highest value possible at 8 bits ($2^8 - 1$). Applying this same tactic to a 16-bit system however didn't seem to work. Pixel values were divided by 65535 ($2^{16} - 1$) and the network was trained. The resulting predicted projections were unusable. For low count projections with pixel values that stayed under 255 the predicted projections were covered in checkerboard artifacts and no longer representative of either the low count or high count projections. For low count projections with pixel values that exceeded 255 the predicted projections blew up, resulting in an image that was almost uniformly high in pixel values. At this point the format inconsistency was given up on and considered a limitation of fast.ai. Tiff files are now opened as 1-channel 16-bit depth images, normalised by dividing by 255, and repeated twice to result in a 3-channel image. As all projections in the training dataset as well as in the test sets stay well below 255 pixel value this is not a problem for this current study, but should this research be continued in the future then users should be aware that projections with pixel values above 255 will cause issues with the current code.

6 Conclusion

The goal of this study was to investigate how to enhance a previously established Perceptual Loss Network such that it could improve low count SPECT projections to high count SPECT projections more accurately. This was done in part by expanding the training set on which the neural networks were trained on, and in part by making adjustments to the neural networks such as switching over to width concatenated projections or supersampling and downsampling the projections during the prediction process.

Reconstructions have shown that there are limitations to the applicability of neural networks. Particularly, very low activity projections are difficult for neural networks to improve. The networks may not be able to extrapolate what kind of structures are meant to be seen in the projections, thus making it very challenging to accurately improve the SPECT projections. It is therefore recommended to expand the training set even further with different very low count projections.

Expanding the training set alone showed very slight improvement in reconstruction CNR, but was in no way world-changing. Applying width concatenation on top of this seemed to improve results a bit more, but it is questionable whether the increase in resource requirements and computing time is worth the marginal increase in CNR. Applying supersampling and downsampling proved to be very promising, increasing the CNR from anywhere between 0.35 to upwards of 0.75 in some cases. It also showed the most potential out of all neural networks examined to improve actual measured SPECT projections. The uniform phantoms had varying results. Very low count uniform cylinder phantoms were able to be improved by neural networks, but did not seem to benefit much from training on the expanded training set. It also seemed that trying to improve higher count uniform cylinder phantoms was difficult for the neural networks as there seem to be artifacts in the neural network reconstructions that decrease uniformity. Taking the results for the uniform cylinder phantoms into account as well one would again come to the conclusion that expanding the training set combined with applying downsampling seems to be the most promising method. Recommendations to further examine and improve this downsampling technique have been provided. It may be interesting for further research to combine the individual improvements performed in this study to see whether they may together offer a better neural network for improving SPECT projections.

References

[1] S. B. Lo, S. A. Lou, Jyh-Shyan Lin, M. T. Freedman, M. V. Chien, and S. K. Mun, "Artificial convolution neural network techniques and applications for lung nodule detection," *IEEE Transactions on Medical Imaging*, vol. 14, no. 4, pp. 711–718, 1995.

[2] M. Khalil, J. Tremoleda, T. Bayomy, and W. Gsell, "Molecular spect imaging: an overview," *International journal of molecular imaging*, vol. 2011, p. 796025, 04 2011.

[3] O. Ivashchenko, F. van der Have, J. L. Villena, H. C. Groen, R. M. Ramakers, H. H. Weinans, and F. J. Beekman, "Quarter-millimeter-resolution molecular mouse imaging with u-spect+," *Molecular Imaging*, vol. 14, no. 1, p. 7290.2014.00053, 2015, pMID: 25429783. [Online]. Available: https://doi.org/10.2310/7290.2014.00053

[4] S. Ramos, S. Gehrig, P. Pinggera, U. Franke, and C. Rother, "Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1025–1032.

[5] G. Iannizzotto, L. L. Bello, A. Nucita, and G. M. Grasso, "A vision and speech enabled, customizable, virtual assistant for smart environments," in *2018 11th International Conference on Human System Interaction (HSI)*, 2018, pp. 50–56.

[6] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 184–199.

[7] F. J. Beekman, F. van der Have, B. Vastenhouw, A. J. van der Linden, P. P. van Rijk, J. P. H. Burbach, and M. P. Smidt, "U-spect-i: a novel system for submillimeter-resolution tomography with radiolabeled molecules in mice," *Journal of Nuclear Medicine*, vol. 46, no. 7, pp. 1194–1200, 2005.

[8] F. Van Der Have, B. Vastenhouw, R. M. Ramakers, W. Branderhorst, J. O. Krah, C. Ji, S. G. Staelens, and F. J. Beekman, "U-spect-ii: an ultra-high-resolution device for molecular small-animal imaging," *Journal of Nuclear Medicine*, vol. 50, no. 4, pp. 599–605, 2009.

[9] "Spect, pet, ct, & optical imaging," May 2020. [Online]. Available: https://www.milabs.com/

[10] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[11] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

[12] J. Howard *et al.*, "fastai," https://github.com/fastai/fastai, 2020.

[13] "Paperspace: Cloud machine learning, ai, and effortless gpu rental." [Online]. Available: https://www.paperspace.com/

[14] J. Johnson, A. Alahi, and F. Li, "Perceptual losses for real-time style transfer and super-resolution," *CoRR*, vol. abs/1603.08155, 2016. [Online]. Available: http://arxiv.org/abs/1603.08155

[15] M. P. Nguyen, M. C. Goorden, C. Kamphuis, and F. J. Beekman, "Evaluation of pinhole collimator materials for micron-resolution ex vivo SPECT," *Physics in Medicine & Biology*, vol. 64, no. 10, p. 105017, may 2019. [Online]. Available: https://doi.org/10.1088%2F1361-6560%2Fab1618

[16] F. van der Have and F. J. Beekman, "Photon penetration and scatter in micro-pinhole imaging: a monte carlo investigation," *Physics in Medicine and Biology*, vol. 49, no. 8, pp. 1369–1386, mar 2004. [Online]. Available: https://doi.org/10.1088%2F0031-9155%2F49%2F8%2F001

[17] S. Jan, G. Santin, D. Strul, S. Staelens, K. Assié, D. Autret, S. Avner, R. Barbier, M. Bardiès, P. M. Bloomfield, D. Brasse, V. Breton, P. Bruyndonckx, I. Buvat, A. F. Chatziioannou, Y. Choi, Y. H. Chung, C. Comtat, D. Donnarieix, L. Ferrer, S. J. Glick, C. J. Groiselle, D. Guez, P.-F. Honore, S. Kerhoas-Cavata, A. S. Kirov, V. Kohli, M. Koole, M. Krieguer, D. J. van der Laan, F. Lamare, G. Largeron, C. Lartizien, D. Lazaro, M. C. Maas, L. Maigne, F. Mayet, F. Melot, C. Merheb, E. Pennacchio, J. Perez, U. Pietrzyk, F. R. Rannou, M. Rey, D. R. Schaart, C. R. Schmidtlein, L. Simon, T. Y. Song, J.-M. Vieira, D. Visvikis, R. V. de Walle, E. Wieërs, and C. Morel, "GATE: a simulation toolkit for PET and SPECT," *Physics in Medicine and Biology*, vol. 49, no. 19, pp. 4543–4561, sep 2004. [Online]. Available: https://doi.org/10.1088%2F0031-9155%2F49%2F19%2F007

[18] S. Jan, D. Benoit, E. Becheva, T. Carlier, F. Cassol, P. Descourt, T. Frisson, L. Grevillot, L. Guigues, L. Maigne, C. Morel, Y. Perrot, N. Rehfeld, D. Sarrut, D. R. Schaart, S. Stute, U. Pietrzyk, D. Visvikis, N. Zahra, and I. Buvat, "GATE v6: a major enhancement of the GATE simulation platform enabling modelling of CT and radiotherapy," *Physics in Medicine and Biology*, vol. 56, no. 4, pp. 881–901, jan 2011. [Online]. Available: https://doi.org/10.1088%2F0031-9155%2F56%2F4%2F001

[19] C.-L. Chen, Y. Wang, J. J. S. Lee, and B. M. W. Tsui, "Integration of simset photon history generator in gate for efficient monte carlo simulations of pinhole spect," *Medical Physics*, vol. 35, no. 7Part1, pp. 3278–3284, 2008. [Online]. Available: https://aapm.onlinelibrary.wiley.com/doi/abs/10.1118/1.2940159

[20] P. E. B. Vaissier, F. J. Beekman, and M. C. Goorden, "Similarity-regulation of OS-EM for accelerated SPECT reconstruction," *Physics in Medicine and Biology*, vol. 61, no. 11, pp. 4300–4315, may 2016. [Online]. Available: https://doi.org/10.1088%2F0031-9155%2F61%2F11%2F4300

[21] H. M. Hudson and R. S. Larkin, "Accelerated image reconstruction using ordered subsets of projection data," *IEEE Transactions on Medical Imaging*, vol. 13, no. 4, pp. 601–609, 1994.

[22] F. van der Have, B. Vastenhouw, M. Rentmeester, and F. J. Beekman, "System calibration and statistical image reconstruction for ultra-high resolution stationary pinhole spect," *IEEE Transactions on Medical Imaging*, vol. 27, no. 7, pp. 960–971, 2008.

[23] MATLAB, *9.7.0.1190202 (R2019b)*. Natick, Massachusetts: The MathWorks Inc., 2018.

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[25] A. Dertat, "Applied deep learning - part 4: Convolutional neural networks," Nov 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2#8efa

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *arXiv preprint arXiv:1804.07612*, 2018.

[30] "Training with mixed precision," Apr 2020. [Online]. Available: https://docs.nvidia.com/deeplearning/performance/pdf/Training-Mixed-Precision-User-Guide.pdf

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] M. D. Walker, M. C. Goorden, K. Dinelle, R. M. Ramakers, S. Blinder, M. Shirmohammad, F. van der Have, F. J. Beekman, and V. Sossi, "Performance assessment of a preclinical pet scanner with pinhole collimation by comparison to a coincidence-based small-animal pet scanner," *Journal of Nuclear Medicine*, vol. 55, no. 8, pp. 1368–1374, 2014.

[33] A. Clark, "Pillow (pil fork) documentation," 2015. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

[34] C. Gohlke, "tifffile." [Online]. Available: https://pypi.org/project/tifffile/

Appendix

A Training set

The training set consists of 11.421 projection pairs from five different simulated phantoms and each of which will be detailed below. All of the training set phantoms were simulated with ^{99m}Tc as radioisotope. The high count projections were generated by taking a random selection of 200/250th of all detection events during the simulation (what will be referred to as 100%), while the low count projections were generated at 20/250th of all detection events (what will be referred to as 10%).

A.1 Derenzo Phantom (6 segments)

This is the part of the training set that PLN Original was trained on. It consists of a Derenzo hot rod phantom with six segments *without* background activity. The segments have rod sizes of 0.17mm, 0.16mm, 0.15mm, 0.14mm, 0.13mm and 0.12mm in diameter and all rods are 1.5mm in length. The centres of the segments are placed at a distance of 0.9mm from the centre axis of the collimator. The activity concentration was set to 4506 MBq/ml during simulation and simulated for 3 hours. This means the high count (100%) projections effectively have an activity concentration of 3600 MBq/ml and the low count (10%) projections 360 MBq/ml. The phantom was simulated *without* background activity. The set consists of 2358 projection pairs. See figure 22 below.

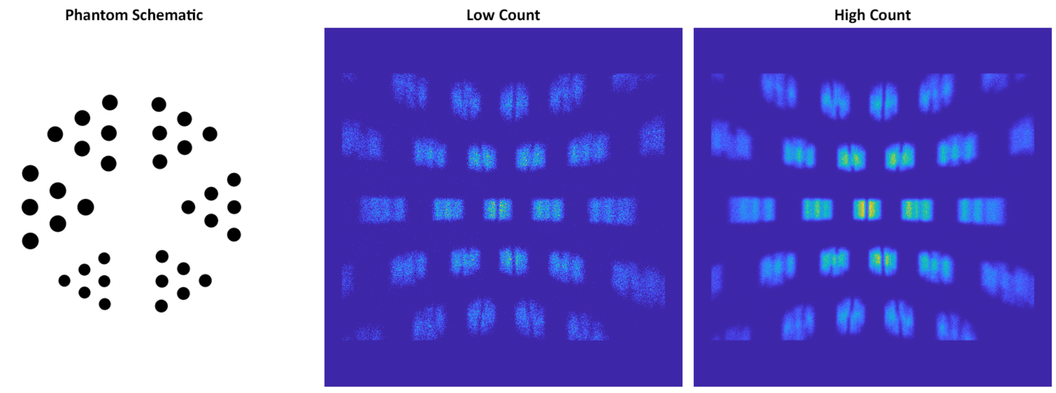


Figure 22: A schematic drawing (left), low count projection (middle) and high count projection (right) of the simulated Derenzo phantom with 6 segments.

A.2 Jaszczak Phantom

A Jaszczak phantom is a type of phantom that does not utilise rods, but instead contains hollow spheres that can be filled with radiopharmaceuticals. The Jaszczak phantom that was simulated has eight spheres placed on a circle 2.4mm in diameter centred at the central axis of the collimator. The diameter of each sphere was 0.64mm, 0.56mm, 0.48mm, 0.42mm, 0.36mm, 0.30mm, 0.24mm and 0.18mm. The activity concentration was set to 4506 MBq/ml during simulation and simulated for 3 hours. This means the high count (100%) projections effectively have an activity concentration of 3600 MBq/ml and the low count (10%) projections 360 MBq/ml. The phantom was simulated *without* background activity. The set consists of 1965 projection pairs. See figure 23 below.

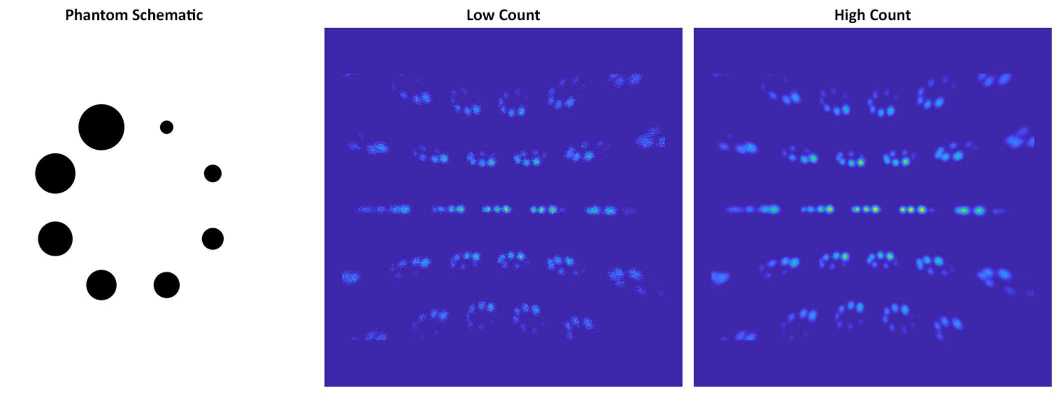


Figure 23: A schematic drawing (left), low count projection (middle) and high count projection (right) of the simulated Jaszczak phantom with 8 spheres.

A.3 Uniform Cylinder Phantom

A uniform cylinder phantom is a type of phantom that just contains one activity volume in the shape of a cylinder. The phantom that was simulated was a cylinder with diameter of 6mm and a length of 1.5mm centred at the central axis of the collimator. The activity concentration was set to 96 MBq/ml during simulation and simulated for 3 hours. This is quite a bit lower than the previous phantoms, but this is compensated for by the large volume of the phantom. This means the high count (100%) projections effectively have an activity concentration of 77 MBq/ml and the low count (10%) projections 7.7 MBq/ml. The phantom was simulated *without* background activity. The set consists of 2586 projection pairs. See figure 24 below.

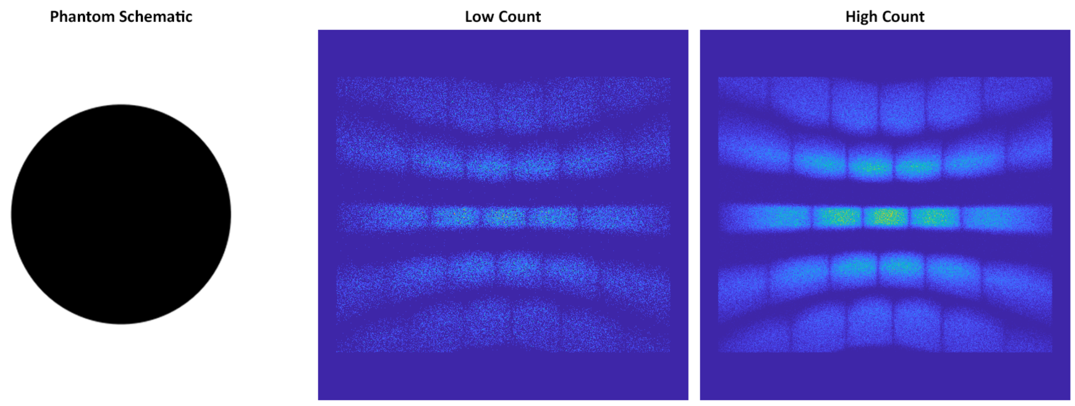


Figure 24: A schematic drawing (left), low count projection (middle) and high count projection (right) of the simulated uniform cylinder phantom.

A.4 Uniform Ellipsoid Phantom + Background

A uniform ellipsoid phantom is a type of phantom similar to the one above, but now the activity volume has the shape of an ellipsoid. The phantom that was simulated is 4mm wide, 2mm high and 4mm long in the direction of the collimator central axis. The ellipsoid was placed in the centre of the collimator. The activity concentration was set to 243 MBq/ml during simulation and simulated for 3 hours. This means the high count (100%) projections effectively have an activity concentration of 194 MBq/ml and the low count (10%) projections 19.4 MBq/ml. The phantom was simulated *with* background activity. The background activity was simulated by creating a volume around the ellipsoid of 10mm diameter, 11mm length, and filling it with 2 MBq/ml activity concentration. The set consists of 2418 projection pairs. See figure 25 below.

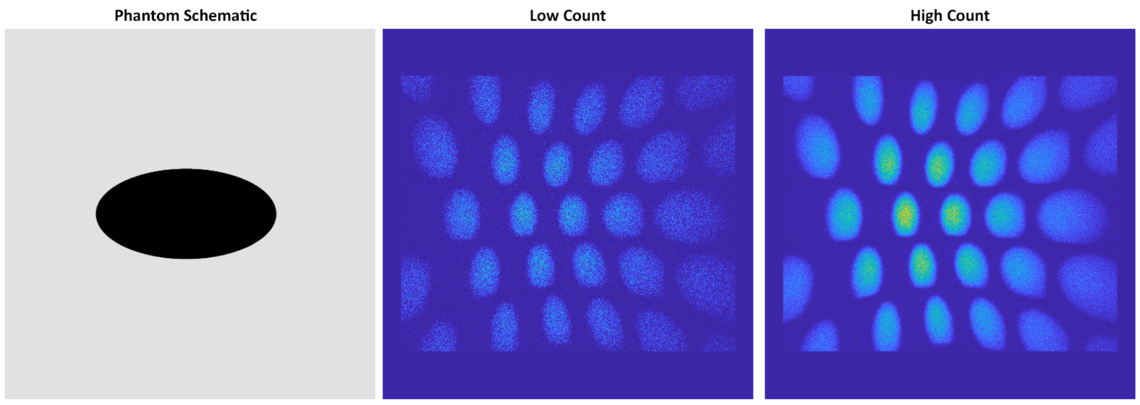


Figure 25: A schematic drawing (left), low count projection (middle) and high count projection (right) of the simulated uniform ellipsoid phantom with background activity.

A.5 Derenzo Phantom (5 segments) + Background

Another Derenzo hot rod phantom was also simulated, this time with 5 segments and *with* background activity. The segments have rod sizes of 0.28mm, 0.24mm, 0.20mm, 0.18mm and 0.16mm in diameter and all rods are 1.5mm in length. The centres of the segments are placed at a distance of 1.2mm from the centre axis of the collimator. The activity concentration was set to 2463 MBq/ml during simulation and simulated for 3 hours. This means the high count (100%) projections effectively have an activity concentration of 1970 MBq/ml and the low count (10%) projections 197 MBq/ml. The phantom was simulated *with* background activity. The background activity was simulated by creating a volume around the Derenzo phantom of 10mm diameter, 11mm length, and filling it with 5 MBq/ml activity concentration. The set consists of 2094 projection pairs. See figure 26 below.

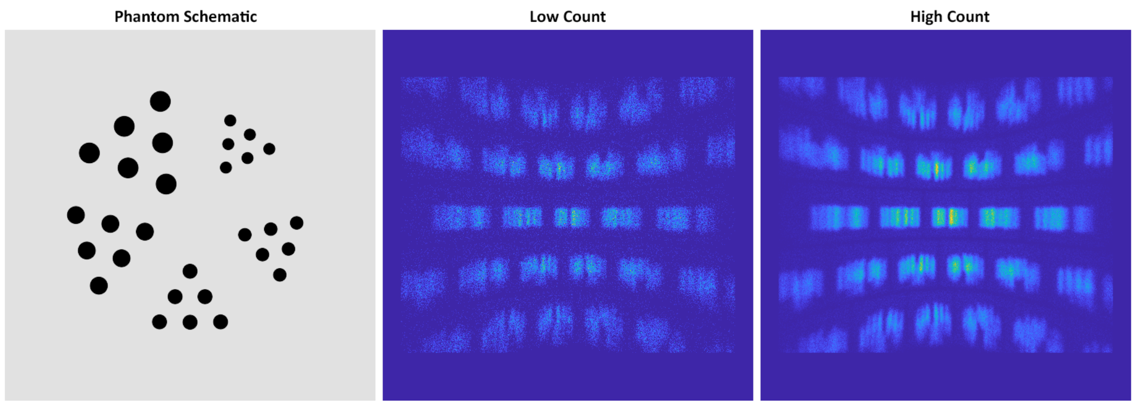


Figure 26: A schematic drawing (left), low count projection (middle) and high count projection (right) of the simulated Derenzo phantom with 5 segments and background activity.

B Test sets

In order to measure the performance of the neural networks several test sets were used. These test sets are both simulated as well as measured and all of the test set phantoms were simulated or measured with ^{99m}Tc as radioisotope.

B.1 Simulated Derenzo Phantom 1

Simulated Derenzo phantom 1 is identical to the projections in subsection A.1 and serves as a control test to see how well the neural networks can improve low count projections that is was directly trained on. The activity concentration of the high count (100%) projections is 3600 MBq/ml and that of the low count (10%) projections is 360 MBq/ml. The phantom was simulated for 3 hours. The set consists of 27 (=3 · 9 bed positions) projections. See figure 27 below.

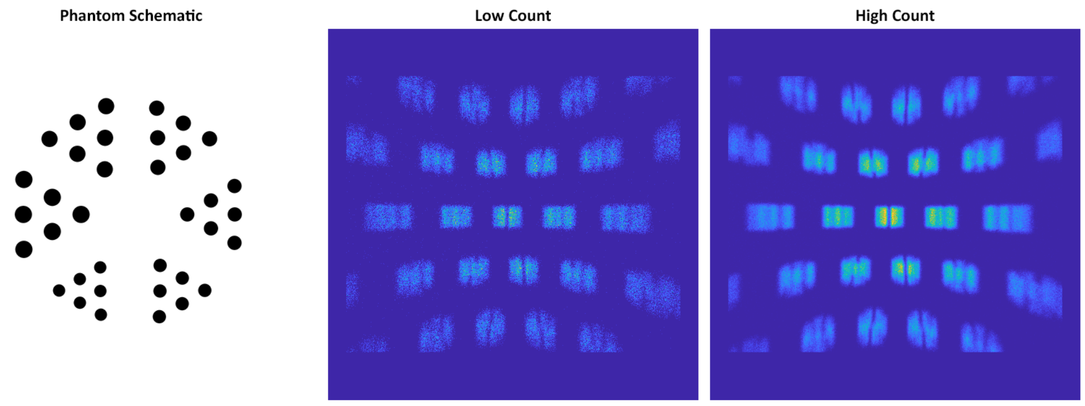


Figure 27: A schematic drawing (left), low count projection (middle) and high count projection (right) of simulated test Derenzo phantom 1.

B.2 Simulated Derenzo Phantom 2

Simulated Derenzo phantom 2 was chosen as a test to see whether the neural networks are able to improve very low activity Derenzo phantom projections that are not directly in the training set. It consists of a Derenzo hot rod phantom with six segments *without* background activity and was simulated for 3 hours. The segments have rod sizes of 0.30mm, 0.26mm, 0.22mm, 0.19mm, 0.17mm and 0.15mm in diameter and all rods are 1.5mm in length. The activity concentration of the high count (100%) projections is 212 MBq/ml and that of the low count (10%) projections is 21.2 MBq/ml. The phantom was simulated for 3 hours. The set consists of 27 (=3 · 9 bed positions) projections. See figure 28 below.

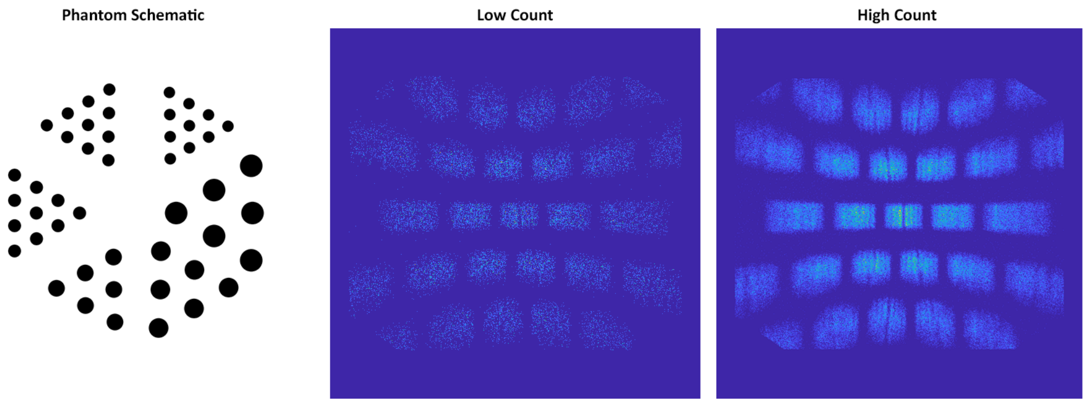


Figure 28: A schematic drawing (left), low count projection (middle) and high count projection (right) of simulated test Derenzo phantom 2.

B.3 Simulated Derenzo Phantom 3

Simulated Derenzo phantom 3 is *identical* to Simulated Derenzo phantom 2. This time however it is the high count projections that are to be improved instead of the low count projections. That is to say the low count projections of simulated Derenzo phantom 3 *are* the high count projections of simulated Derenzo phantom 2. This also means that there will be no high count projections for this test set to compare against. The activity concentration of the low count is thus 212 MBq/ml. The phantom was simulated for 3 hours. The set consists of 27 (=3 · 9 bed positions) projections. See figure 29 below.

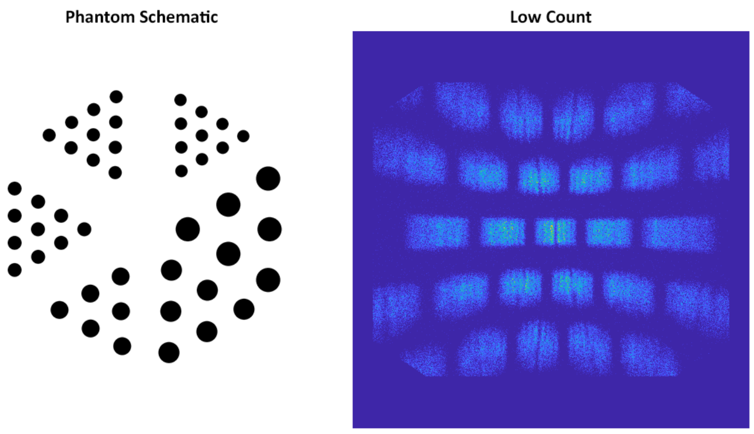


Figure 29: A schematic drawing (left) and low count projection (right) of simulated test Derenzo phantom 3.

B.4 Measured Derenzo Phantom

The measured Derenzo phantom was added as a test set to see whether the neural networks can improve SPECT projections that are physically measured as opposed to simulated. This was considered an interesting test since real phantoms are often a bit messy. The radiopharmaceutical fluid is often difficult to clean off of areas where there is supposed to be none, so there is usually some residual fluid left on the edges of the phantom also emitting radiation that is picked up by the detector plates. Additionally, the effects of scatter are also included in these projections because this phantom was physically measured. It consists of a Derenzo hot rod phantom with six segments. The segments have rod sizes of 0.17mm, 0.16mm, 0.15mm, 0.14mm, 0.13mm and 0.12mm in diameter and all rods are 1.5mm in length. The activity concentration of the high count (100%) projections is 4510 MBq/ml and that of the low count (10%) projections is 451 MBq/ml. The phantom was measured for 3 hours. The set consists of 27 (=3 · 9 bed positions) projections. See figure 30 below.

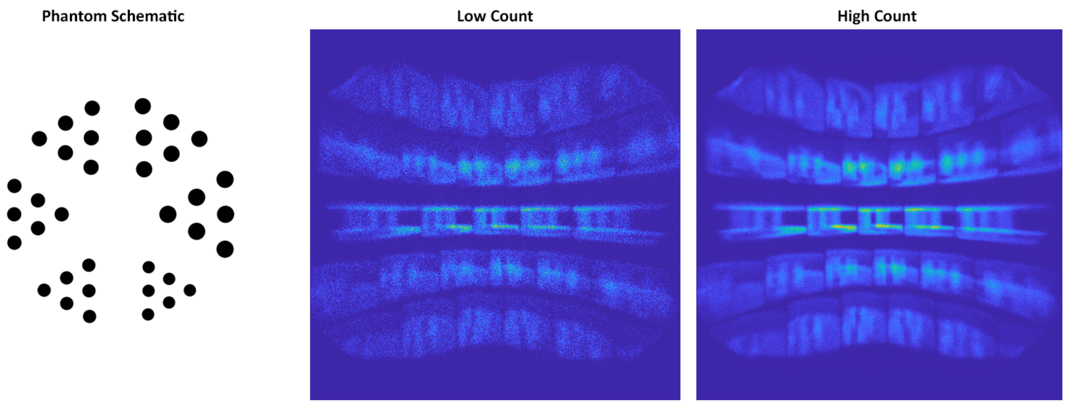


Figure 30: A schematic drawing (left), low count projection (middle) and high count projection (right) of the measured test Derenzo phantom.

B.5 Simulated Uniform Cylinder 1

A simulated uniform phantom was added as a test set specifically to see whether the neural networks had benefited from training on additional types of phantoms when trying to improve non-Derenzo phantoms. No details were specified for this phantom (such as activity concentration or simulation time). Its only purpose is to see whether the neural networks can improve the projections. The phantom was simulated *without* background activity. The set consists of 135 (=3 · 45 bed positions) projections. See figure 31 below.

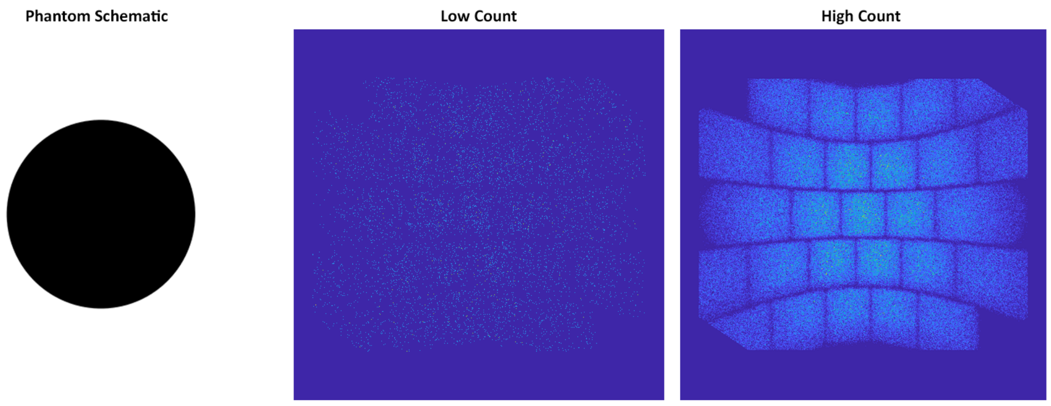


Figure 31: A schematic drawing (left), low count projection (middle) and high count projection (right) of simulated uniform cylinder phantom 1.

B.6 Simulated Uniform Cylinder 2

Simulated uniform phantom 2 is *identical* to simulated Derenzo phantom 1. This time however it is the high count projections that are to be improved instead of the low count projections. That is to say the low count projections of simulated uniform phantom 2 *are* the high count projections of simulated uniform phantom 1. This also means that there will be no high count projections for this test set to compare against. Once again no details were specified for this phantom (such as activity concentration or simulation time). The phantom was simulated *without* background activity. The set consists of 135 (=3 · 45 bed positions) projections. See figure 32 below.

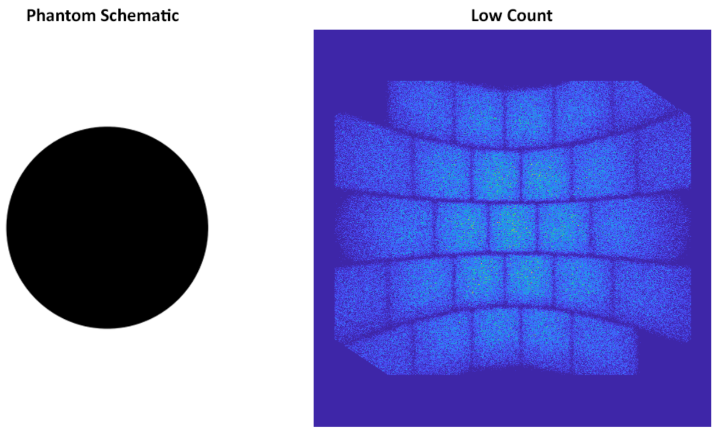


Figure 32: A schematic drawing (left) and low count projection (right) of simulated uniform cylinder phantom 2.

B.7 Measured Uniform Cylinder

As the simulated uniform phantom was quite low activity a measured uniform cylinder phantom was also added to the dataset with a higher activity. The measured phantom was a cylinder with diameter of 8.5mm and a length of 9.5mm centred at the central axis of the collimator. The high count projections were unfortunately unavailable, so high count reconstructions are left out of the results as well. The activity concentration of the high count (100%) projections is 38.6 MBq/ml and that of the low count (10%) projections is 3.86 MBq/ml. The phantom was measured for 3 hours. The set consists of 384 (=$3 \cdot 128$ bed positions) projections. See figure 33 below.

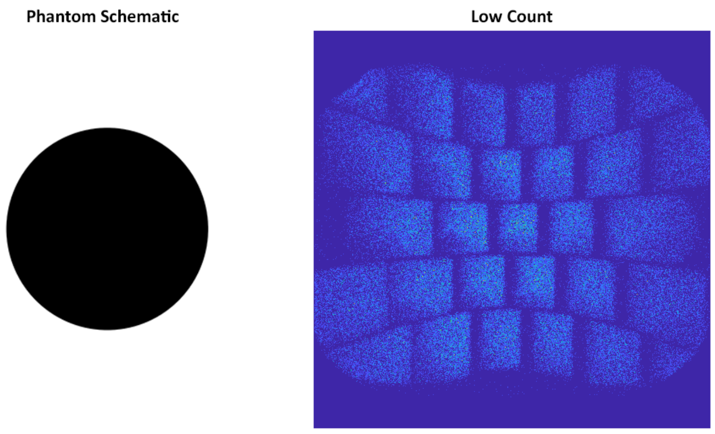


Figure 33: A schematic drawing (left) and low count projection (right) of the measured uniform cylinder phantom.

C Jupyter Notebook of PLN Expanded

Initialisation

```
%matplotlib inline
from fastai.basics import *
from fastai.vision import *
import fastai
from fastai.callbacks import *
from fastai.vision.gan import *
from fastai.vision.learner import cnn_config
import torch.nn as nn
import random
import os
import torchvision
from torchvision.models import vgg19_bn
import tifffile
```

```
# Establish paths
pathroot = Path('D:/BEP/proj_20_200')
path = pathroot/'data'
path_tests = pathroot/'tests'
path_models = pathroot/'models'
path_predictions = pathroot/'predictions'
```

```
# Clean the training dataset.
# Any projections that do not form a low-high pair will be removed from the dataset.
data_paths = os.listdir(path/'high_count')
for data_path in data_paths:
    high_images = get_image_files(path/'high_count'/data_path)
    low_images = get_image_files(path/'low_count'/data_path)

    high_str = [0]*len(high_images)
    for i in  range(0,len(high_images)):
        high_str[i] = os.path.basename(high_images[i]);
    low_str = [0]*len(low_images)
    for i in  range(0,len(low_images)):
        low_str[i] = os.path.basename(low_images[i]);
    common_str = [x for x in low_str if x in high_str]

    for i in  range(0,len(high_images)):
        if os.path.basename(high_images[i]) not in common_str:
            os.remove(high_images[i])
    for i in  range(0,len(low_images)):
        if os.path.basename(low_images[i]) not in common_str:
            os.remove(low_images[i])

    high_images = get_image_files(path/'high_count'/data_path)
    low_images = get_image_files(path/'low_count'/data_path)
    print('Length of high count projections in ' + data_path + ' is:', len(high_images))
    print('Length of low  count projections in ' + data_path + ' is:', len(low_images))
```

Create custom classes for Tiff handling

```
# Creates a custom open function based on the open_image() function provided by fast.ai.
# Allows for proper loading in of the 16-bit 512x512 grayscale tiff images.
# Returns an Image object created from the image at fn.
def open_tiff_image(fn:PathOrStr, div:bool=True, cls:type=Image, after_open:Callable=None)->Image:
    x = tifffile.imread(str(fn), key=0)
    if after_open: x = after_open(x)
    x = pil2tensor(x,np.float32)
    if div: x.div_(255)
    x = np.repeat(x, 3, axis=0)
    return cls(x)
```

```python
# Creates a custom TiffImageList class based on the ImageList class provided by fast.ai.
# This is identical to ImageList apart from the open() function,
# which now uses the open_tiff_image() function defined above.
image_extensions = set(k for k,v in mimetypes.types_map.items() if v.startswith('image/'))
class TiffImageList(ItemList):
    _bunch,_square_show,_square_show_res = ImageDataBunch,True,True
    def __init__(self, *args, convert_mode='RGB', after_open:Callable=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.convert_mode,self.after_open = convert_mode,after_open
        self.copy_new += ['convert_mode', 'after_open']
        self.c,self.sizes = 3,{}

    def open(self, fn):
        return open_tiff_image(fn, after_open=self.after_open)

    def get(self, i):
        fn = super().get(i)
        res = self.open(fn)
        self.sizes[i] = res.size
        return res

    @classmethod
    def from_folder(cls, path:PathOrStr='.', extensions:Collection[str]=None, **kwargs)->ItemList:
        extensions = ifnone(extensions, image_extensions)
        return super().from_folder(path=path, extensions=extensions, **kwargs)

    @classmethod
    def from_df(cls, df:DataFrame, path:PathOrStr, cols:IntsOrStrs=0, folder:PathOrStr=None,
                suffix:str='', **kwargs)->'ItemList':
        suffix = suffix or ''
        res = super().from_df(df, path=path, cols=cols, **kwargs)
        pref = f'{res.path}{os.path.sep}'
        if folder is not None: pref += f'{folder}{os.path.sep}'
        res.items = np.char.add(np.char.add(pref, res.items.astype(str)), suffix)
        return res

    @classmethod
    def from_csv(cls, path:PathOrStr, csv_name:str, header:str='infer', delimiter:str=None,
                **kwargs)->'ItemList':
        path = Path(path)
        df = pd.read_csv(path/csv_name, header=header, delimiter=delimiter)
        return cls.from_df(df, path=path, **kwargs)

    def reconstruct(self, t:Tensor): return Image(t.float().clamp(min=0,max=1))

    def show_xys(self, xs, ys, imgsize:int=4, figsize:Optional[Tuple[int,int]]=None, **kwargs):
        rows = int(np.ceil(math.sqrt(len(xs))))
        axs = subplots(rows, rows, imgsize=imgsize, figsize=figsize)
        for x,y,ax in zip(xs, ys, axs.flatten()): x.show(ax=ax, y=y, **kwargs)
        for ax in axs.flatten()[len(xs):]: ax.axis('off')
        plt.tight_layout()

    def show_xyzs(self, xs, ys, zs, imgsize:int=4, figsize:Optional[Tuple[int,int]]=None, **kwargs):
        if self._square_show_res:
            title = 'Ground truth\nPredictions'
            rows = int(np.ceil(math.sqrt(len(xs))))
            axs = subplots(rows, rows, imgsize=imgsize, figsize=figsize, title=title, weight='bold', size=12)
            for x,y,z,ax in zip(xs,ys,zs,axs.flatten()): x.show(ax=ax, title=f'{str(y)}\n{str(z)}', **kwargs)
            for ax in axs.flatten()[len(xs):]: ax.axis('off')
        else:
            title = 'Ground truth/Predictions'
            axs = subplots(len(xs), 2, imgsize=imgsize, figsize=figsize, title=title, weight='bold', size=14)
            for i,(x,y,z) in enumerate(zip(xs,ys,zs)):
                x.show(ax=axs[i,0], y=y, **kwargs)
                x.show(ax=axs[i,1], y=z, **kwargs)
```

```
# Creates a custom TiffImageImageList class based on the ImageImageList class provided by fast.ai.
# This is identical to ImageImageList, but it now depends on TiffImageList instead of ImageList.
class TiffImageImageList(TiffImageList):
    _label_cls,_square_show,_square_show_res = TiffImageList,False,False

    def show_xys(self, xs, ys, imgsize:int=4, figsize:Optional[Tuple[int,int]]=None, **kwargs):
        axs = subplots(len(xs), 2, imgsize=imgsize, figsize=figsize)
        for i, (x,y) in enumerate(zip(xs,ys)):
            x.show(ax=axs[i,0], **kwargs)
            y.show(ax=axs[i,1], **kwargs)
        plt.tight_layout()

    def show_xyzs(self, xs, ys, zs, imgsize:int=4, figsize:Optional[Tuple[int,int]]=None, **kwargs):
        title = 'Input / Prediction / Target'
        axs = subplots(len(xs), 3, imgsize=imgsize, figsize=figsize, title=title, weight='bold', size=14)
        for i,(x,y,z) in enumerate(zip(xs,ys,zs)):
            x.show(ax=axs[i,0], **kwargs)
            y.show(ax=axs[i,2], **kwargs)
            z.show(ax=axs[i,1], **kwargs)
```

Create the Databunch

```
# Initialises the batch size, image size, U-Net encoder (arch) and the dataset
# Image size is decreased to 256x256 at first to allow the learner tot learn general/larger structures
bs,size=8,512
bs,size=bs*2,size//2
arch = models.resnet34
src = TiffImageImageList.from_folder(path/'low_count', convert_mode='I;16').split_by_rand_pct(0.1, seed=42)
```

```
# Initialises functions necessary to create the databunch properly.
# The correct_path() function is dependent on the user's individual setup.
def correct_path(path_in):
    y = str(path_in).split('\\')
    y[4] = 'high_count'
    y = '\\'.join(y)
    return Path(y)

def get_data(bs,size):
    data = (src.label_from_func(lambda x: correct_path(x), convert_mode='I;16')
            .transform(get_transforms(max_zoom=2.,max_rotate=15), size=size, tfm_y=True)
            .databunch(bs=bs, num_workers=0))
    data.c = 3
    return data
```

```
# Creates the databunch with the desired batch size and image size
data = get_data(bs,size)
```

Set up Feature Loss

```
def gram_matrix(x):
    n,c,h,w = x.size()
    x = x.view(n, c, -1)
    return (x @ x.transpose(1,2))/(c*h*w)
```

```
loss_m = vgg19_bn(True).features.cuda().eval()
requires_grad(loss_m, False)
blocks = [i-1 for i,o in enumerate(children(loss_m)) if isinstance(o,nn.MaxPool2d)]
base_loss = F.l1_loss
```

```
class FeatureLoss(nn.Module):
    def __init__(self, m_feat, layer_ids, layer_wgts):
        super().__init__()
        self.m_feat = m_feat
        self.loss_features = [self.m_feat[i] for i in layer_ids]
        self.hooks = hook_outputs(self.loss_features, detach=False)
        self.wgts = layer_wgts
        self.metric_names = ['pixel',] + [f'feat_{i}' for i in range(len(layer_ids))
              ] + [f'gram_{i}' for i in range(len(layer_ids))]

    def make_features(self, x, clone=False):
        self.m_feat(x)
        return [(o.clone() if clone else o) for o in self.hooks.stored]

    def forward(self, input, target):
        out_feat = self.make_features(target, clone=True)
        in_feat = self.make_features(input)
        self.feat_losses = [base_loss(input,target)]
        self.feat_losses += [base_loss(f_in, f_out)*w
                             for f_in, f_out, w in zip(in_feat, out_feat, self.wgts)]
        self.feat_losses += [base_loss(gram_matrix(f_in), gram_matrix(f_out))*w**2 * 5e3
                             for f_in, f_out, w in zip(in_feat, out_feat, self.wgts)]
        self.metrics = dict(zip(self.metric_names, self.feat_losses))
        return sum(self.feat_losses)

    def __del__(self): self.hooks.remove()
```

```
feat_loss = FeatureLoss(loss_m, blocks[2:5], [5,15,2]).cuda()
```

Train the network

```
# Initialise the U-Net learner, set learner to use half precision and send learner to the GPU.
wd = 1e-3
learn = unet_learner(data, arch, wd=wd, loss_func=feat_loss, callback_fns=LossMetrics,
                     blur=True, self_attention=True, norm_type=NormType.Weight)
learn.to_fp16();
learn.model.cuda();
```

```
# Prints a summary of the learner
learn.summary()
```

```
# Plots the loss as a function of the learn rate
learn.lr_find()
learn.recorder.plot()
```

```
# Define do_fit() function used to train the learner
lr = 1e-4
def do_fit(save_name, no_of_cycles, lrs=slice(lr), pct_start=0.9):
    learn.fit_one_cycle(no_of_cycles, lrs, pct_start=pct_start)
    learn.save(save_name)
    learn.show_results(rows=1, imgsize=5)
```

```
do_fit('PLN_Expanded_1a', 10, slice(lr*10))
```

```
learn.unfreeze()
do_fit('PLN_Expanded_1b', 10, slice(1e-5,lr))
```

```
# Increase image size to 512x512, decrease batch size to keep GPU memory usage down
data = get_data(bs//2,size*2)
learn.data = data;
learn.to_fp16();
learn.freeze();
```

```
do_fit('PLN_Expanded_2a', 10)
```

```
learn.unfreeze()
do_fit('PLN_Expanded_2b', 10, slice(1e-6,1e-4), pct_start=0.3)
```

```
do_fit('PLN_Expanded_2c', 10, slice(1e-6,1e-4), pct_start=0.3)
```

Perform predictions on test sets

```
# Load trained model to produce predictions
learn.load(path_models/'PLN_Expanded_2c');
learn.data = data;
learn.to_fp32();
```

```
# Create folder to store predictions
name_gen = 'PLN_Expanded_test_same_phantom_prediction'
path_gen = path_predictions/name_gen
path_gen.mkdir(exist_ok=True)
```

```
# Define function to produce predictions
# Uses special save function that saves predictions as 16-bit 512x512 grayscale images.
def save_preds(dl):
    def uint16_save(self, fn:PathOrStr):
        x = image2np(self.data*255).astype(np.uint16)
        tifffile.imsave(fn, x)
    i=0
    names = dl.dataset.items
    for b in dl:
        preds = learn.pred_batch(batch=b, reconstruct=True)
        for o in preds:
            uint16_save(o, path_gen/names[i].name)
            i += 1
```

```
# Define function to produce predictions
# Uses special save function that saves predictions as 16-bit 512x512 grayscale images.
def save_preds(dl):
    def uint16_save(self, fn:PathOrStr):
        x = image2np(self.data*255).astype(np.uint16)
        tifffile.imsave(fn, x)
    i=0
    names = dl.dataset.items
    for b in dl:
        preds = learn.pred_batch(batch=b, reconstruct=True)
        for o in preds:
            uint16_save(o, path_gen/names[i].name)
            i += 1
```

```
# Load in test projections, create databunch and produce predictions
src_test = TiffImageImageList.from_folder(path_tests/'test_same_phantom', convert_mode='I;16').split_none()
data_test = src_test.label_from_func(lambda x: path_tests/'test_same_phantom'/x.name, convert_mode='I;16')
            .databunch(bs=bs, num_workers=0)
save_preds(data_test.fix_dl)
```