Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Bioingegneria

Deep Learning for PET Imaging

from Denoising to Learned Primal-Dual Reconstruction

Relatore:

Prof. Alessandra Bertoldo

Laureando:

Alessandro Guazzo

Anno accademico 2019/2020

Deep Learning for PET Imaging: from Denoising to Learned Primal-Dual Reconstruction

ALESSANDRO GUAZZO

Master in Medical Engineering
Date: April 8, 2020
Supervisor: Massimiliano Colarieti-Tosti
Examiner: Matilda Larsson
School of Engineering Sciences in Chemistry, Biotechnology and Health
Swedish title: Djupinlärning i PET-avbildning: från brusreducering till Learned Primal-Dual Bildrekonstruktion

Abstract

PET imaging is a key tool in the fight against cancer. One of the main issues of PET imaging is the high level of noise that characterizes the reconstructed image, during this project we implemented several algorithms with the aim of improving the reconstruction of PET images exploiting the power of Neural Networks. First, we developed a simple denoiser that improves the quality of an image that has already been reconstructed with a reconstruction algorithm like the Maximum Likelihood Expectation Maximization. Then we implemented two Neural Network based iterative reconstruction algorithms that reconstruct directly an image starting from the measured data rearranged into sinograms, thus removing the dependence of the reconstruction result from the initial reconstruction needed by the denoiser. Finally, we used the most promising approach, among the developed ones, to reconstruct images from data acquired with the KTH MTH microCT - miniPET.

Sammanfattning

PET-avbildning är ett viktigt verktyg i kampen mot cancer. En av huvudfrågorna för PET-avbildning är den höga brusnivån som kännetecknar den rekonstruerade bilden, under detta projekt implementerade vi flera algoritmer i syfte att förbättra återuppbyggnaden av PET-bilder som utnyttjar kraften i Neural Networks. Först utvecklade vi en enkel denoiser som förbättrar kvaliteten på en bild som redan har rekonstruerats med en rekonstruktionsalgoritm som Maximization of Maximum Likelihood Expectation Maximization. Sedan implementerade vi två neurala nätverksbaserade iterativa rekonstruktionsalgoritmer som rekonstruerar direkt en bild med utgångspunkt från de uppmätta data som är omordnade till sinogram och därmed avlägsnar beroendet av rekonstruktionsresultatet från den ursprungliga rekonstruktionen som krävs av deniseraren. Slutligen använde vi det mest lovande tillvägagångssättet, bland de utvecklade, för att rekonstruera bilder från data som skaffats med KTH MTH microCT - miniPET.

Acknowledgements

Many are the people that i need to thank for the success of this project, and I will do that in order of appearance. I will start by thanking my **Parents** for the continuous support from the start to the end of my Swedish adventure. I then thank my supervisor **Massimiliano Colarieti-Tosti** (mamo) for giving me the chance to work on such an interesting and new topic. Thank you to **Andrés Martínez Mora** for starting the development of some code and helping me to get familiar with the topic, an then to **Ozan Öktem** for making me believe more in my ideas than in those of the others. A big thank you to **Fabian Sinzinger** and **Daniel Jörgens** for solving the frequent problems with the "Beast" server. Thank you **Olivier Verdier** for the very interesting discussions on the project topic and to my reviewer **Rodrigo Moreno** for the many suggestions that helped me improve the global quality of my thesis text. I also need to say thank you to **Peter Arfert** for the help in the design and print process of the miniPET phantoms and to all the people of the **Nuclear Medicine KS Huddinge** for giving us some radioactive FDG even during Fika time, the last part of this project would not have been possible without you. Finally I want to thank all the new **Friends** that I met here in Stockholm during my stay for all the happy moments we shared together.

Stockholm, January 2020
Alessandro Guazzo

Contents

"Per Aspera Sic Itur Ad Astra"

Chapter 1

Introduction

Two are the main topics that characterize this project: *Positron Emission Tomography* (PET) imaging and *Deep Learning* (DL).

PET imaging is being increasingly used for diagnosis of various malignancies. Other imaging techniques as *Computed Tomography* (CT) or *Magnetic Resonance* (MR) rely on anatomic changes for diagnosis of cancer. However, PET has the ability to demonstrate abnormal metabolic activity in organs that at a given stage do not show an abnormal appearance based on morphologic criteria. PET is also useful in the follow-up of patients following chemotherapy or surgical resection of tumor, most of whom have a complicating appearance at CT or MR imaging due to postoperative changes or scar tissue. PET imaging is thus a key tool in the fight against cancer.

On the other hand DL is a branch of *Machine Learning* (ML) which is completely based on *Artificial Neural Networks* (NN) and has been one of the trending topics of information engineering research in the last few years. Many different algorithms rely on NN to solve complex problems thanks to their great flexibility and high potential, most of these algorithms employs NN to solve classification or segmentation problems but recently they are starting to be used also for signal or image denoising and reconstruction.

The first part of this master thesis will focus on the development of a NN based denoising algorithm for PET images, then the focus will shift towards reconstruction problems where NN based reconstruction algorithms originally developed for CT data will be rethought and adapted in the PET imaging context. Finally, NN based reconstruction algorithms are going to be used in order to reconstruct images from data acquired with the KTH MTH microCT - miniPET.

Chapter 2

Methods

In this chapter the methods used to obtain the results are described. First denoising and reconstruction problems are introduced in a formal fashion, we then continue with the explanation of how synthetic and miniPET training and test data are generated. The architectures of the CNNs for both denoising and reconstruction problems are also described and finally we explain the training procedures used to train the different networks and how their performance have been evaluated.

2.1 Denoising and Reconstruction Problems

Two different approaches have been considered in order to improve the quality of PET images using deep learning:

- Denoising: The denoising approach is an image to image method since both the network input and output belong to the same space, the image space X_{im}. This means that it is necessary to use an algorithm to reconstruct the image $f \in X_{\text{im}}$ starting from the measured data $s \in X_{\text{d}}$, where X_{d} is the data space. In this project we used one iteration of the MLEM algorithm (MLEM_1) to obtain the input image for the network. The reconstructed image is then given as input to the network Λ that gives as a result the denoised image x_{den}. The mathematical operator describing this approach is thus:

$$x_{\text{den}} = \Lambda(\text{MLEM}_1(s)) \tag{2.1}$$

  We decided to use the denoising term to describe this approach since we give as input to the network an image that is very noisy and we obtain, as

an output, an image that is almost noise free. However, the network will not just remove the noise but also increase the contrast and improve the dynamic range, results that are typically obtained with a sharpening operation. Denoising is thus not the only term that can be used to describe the effect of the network on the input image.

- Reconstruction: The reconstruction approach is a data to image method since the neural network-based iterative reconstruction algorithm $\mathcal{A}$ input is the sinogram obtained from the measured data $s \in X_{\text{d}}$ but its output is the reconstructed image $x_{\text{rc}} \in X_{\text{im}}$.

$$x_{\text{rc}} = \mathcal{A}(s) \tag{2.2}$$

2.2 Synthetic Data Generation

2.2.1 Ground Truth Images Generation

The ground truth images set Y consists of images of a random number of randomly distributed ellipses with random size and intensity. Overlap between different ellipses is allowed. The size of such images is 147×147, the same size of images that can be acquired with the miniPET of the KTH laboratory.

The ground truth images set is generated with three functions developed with the *Python* programming language:

- `generate_ellipsoids_2d`: A function that generates an image such as those described above, the number of ellipses n is extracted from a Poisson distribution with $\lambda = 20$. The intensity of an ellipse is extracted from an uniform distribution between 0 and 1, the coordinates of the center are extracted from an uniform distribution between 0 and 147, the axis lengths are extracted from an exponential distribution with $\lambda = 2$. Once all parameters have been extracted, an ellipse is generated with the *Operator Discretization Library Documentation* (ODL) function `odl.phantom.geometric.ellipsoid_phantom`. This process is repeated n times and the generated ellipses are then placed together in the same image.

- `ellipse_batch`: A function that uses `generate_ellipsoids_2d` to generate a batch of ground truth images, the batch size is a parameter that is given as input to the function,

- `RandomEllipsoids`: A function that generates the whole ground truth images set using the `ellipse_batch` function, the total number of images to be generated is a parameter that is given as input to the function.

Figure 2.1 shows an example of ground truth images generated as described above.

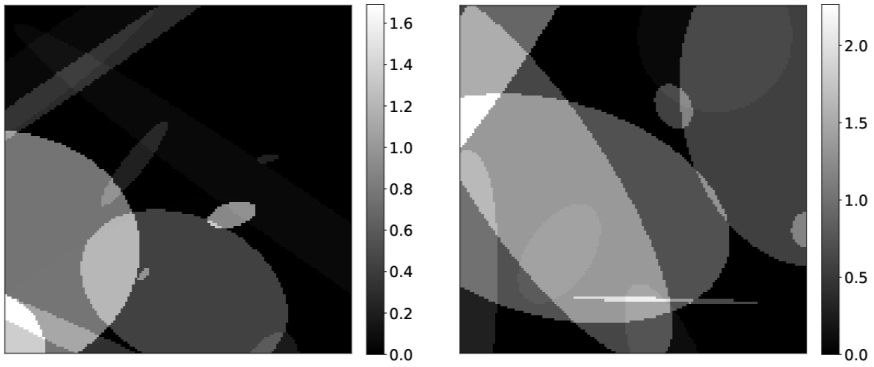


Figure 2.1: Ground truth images

2.2.2 Network Input Images Generation

The network input images set is different for denoising and reconstruction problems. In denoising problems the network input images set X_{dn} consists of images reconstructed with one iteration of the MLEM algorithm, MLEM_1, starting from noisy sinograms obtained from the ground truth images set. For each image in Y we obtain the corresponding image in X_{dn} with the following functions implemented with *Python* programming language.

- `fwd_op_mod`: A function that uses some basic ODLPET functions to compute the sinogram of an image given as input. The number of projections is set to 180 and the number of bins set to 147, equal to the size of the images. These dimensions are the same of a sinogram obtained with the miniPET of the KTH laboratory.

- `generate_data`: A function that uses `fwd_op_mod` to compute the sinogram and then produces a noisy version of it. The noisy version of the sinogram is obtained by extracting every value of the sinogram from

a Poisson distribution with λ equal to the noise free sinogram value divided by the noise level, after each value has been obtained the resulting image is multiplied by the noise level,

- `mlem_op_net`: A function that given a sinogram as input computes the corresponding MLEM_1 reconstruction. The result is achived using the `odl.solvers.iterative.statistical.mlem` ODL function $n_{\text{iter}} = 1$.

Figure 2.2 shows an example of denoising network input images obtained as described above.

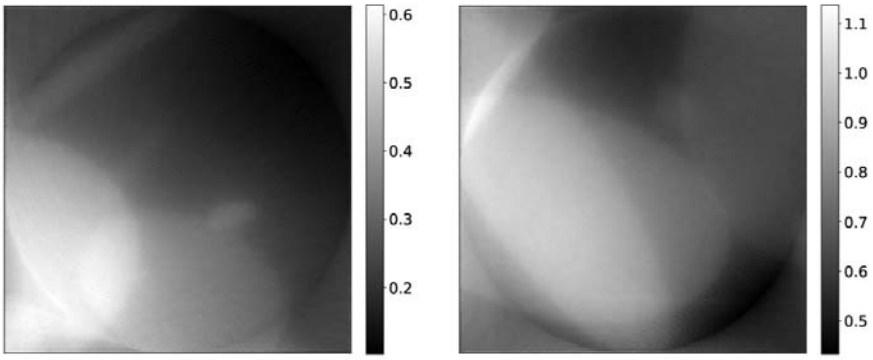


Figure 2.2: Denoising network input images

In reconstruction problems the network input images set X_{rc} consists of noisy sinograms obtained with the `generate_data` function. Figure 2.3 shows an example of reconstruction network input images.

An element in the training set for denoising problems S_{dn} consists of a reconstructed image $x \in X_{dn}$ and the corresponding ground truth image $y \in Y$, as shown in Figure 2.4. An element in the training set for reconstruction problems S_{rc} consists of a noisy sinogram $s \in X_{rc}$ and the corresponding ground truth image $y \in Y$, as shown in Figure 2.5.

The level of noise for the train data can be set in two different ways:

- Fixed noise level: The noise level of the train data is set equal to $\frac{1}{3}$ for all the images. This value has been chosen in order to generate train images that have a similar noise level to images acquired with the miniPET. In order to obtain the noise level we select an uniform region of a MLEM_{10} reconstruction of miniPET data and compute the *Coefficient of Variation*

(CV) then compare this value with the one obtained from a uniform region of a MLEM_{10} reconstruction of noisy data obtained from a ground truth image,

- Variable noise level: The noise level of the train images is independently extracted from an uniform distribution between $\frac{1}{10}$ and $\frac{1}{3}$ for each image. The two extremes of the distribution have been obtained considering the variability in the noise levels of miniPET data and by observing that MLEM_{10} reconstructions of miniPET data and simulated data with noise levels in this interval lead to similar results when the images contain similar objects. Noise levels are extracted from an uniform distribution since we want that each noise level, inside the fixed interval, has the same probability of being extracted thus leading to high noise variability.

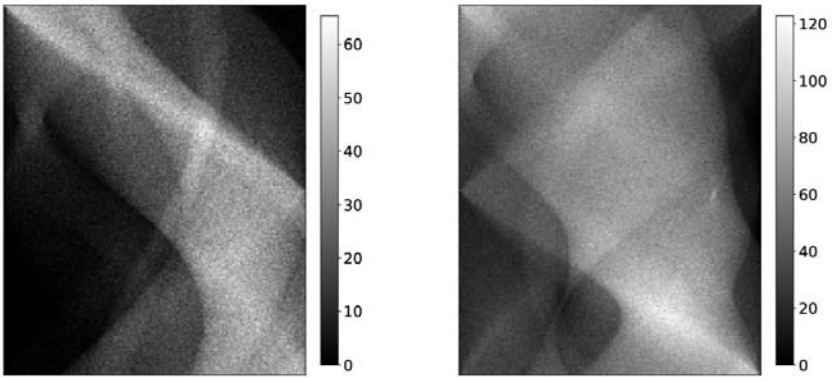


Figure 2.3: Reconstruction network input images

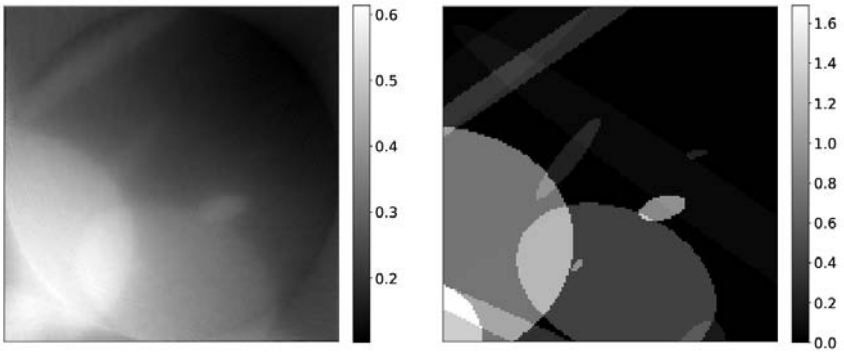


Figure 2.4: Denoising training set element

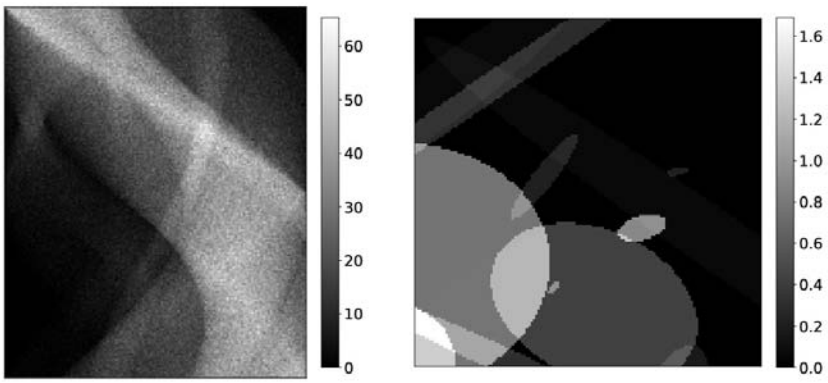


Figure 2.5: Reconstruction training set element

2.2.3 Test Images Generation

The test images set Z consists of 77 slices of size 147×147 from the *Shepp-Logan Phantom* (SLP). Figure 2.6 shows an example of test images.

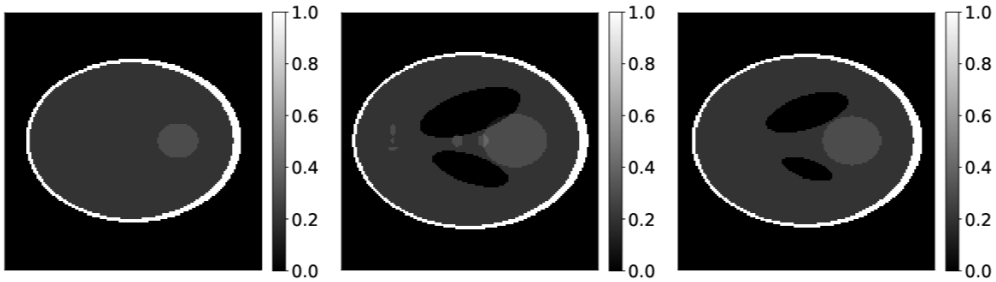


Figure 2.6: Test images

These images are NOT included in the training set and are only used to obtain a noisy sinogram with the `generate_data` function. Starting from the sinograms obtained from the test images a ten iteration MLEM reconstruction, MLEM_{10}, is performed in order to obtain benchmark images to be compared against the network output. Figure 2.7 shows an example of a niosy sinogram, a test image and the corresponding MLEM_{10} reconstruction. The MLEM_{10} reconstruction is computed with the `mlem_op_comp` function that given a sinogram as input computes the corresponding MLEM_{10} reconstruction. The `odl.solvers.iterative.statistical.mlem` ODL function with $n_{\text{iter}} = 10$ is used to achieve this result.

For denoising problems the noisy sinogram obtained from the test images set is used to obtain the MLEM_1 reconstruction that is then given as input to

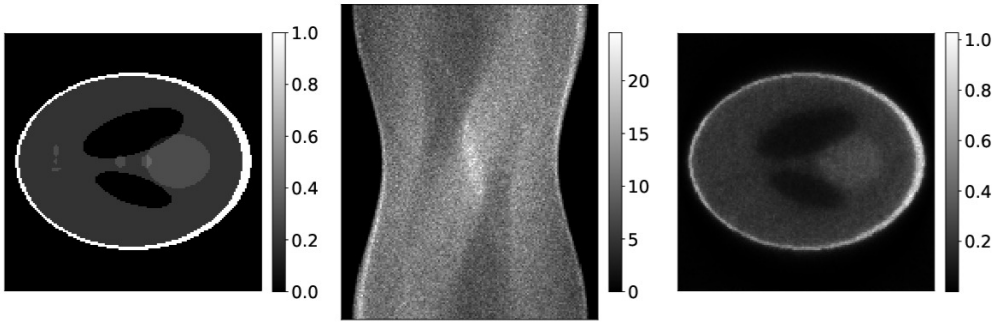


Figure 2.7: Test image, noisy sinogram and MLEM_{10} reconstruction

the denoising networks. Figure 2.8 shows some examples of MLEM_1 reconstruction of test images.

For reconstruction problems the noisy sinogram obtained from the test images is given as input to the reconstruction networks. Figure 2.9 shows some examples of noisy sinograms obtained from test images.

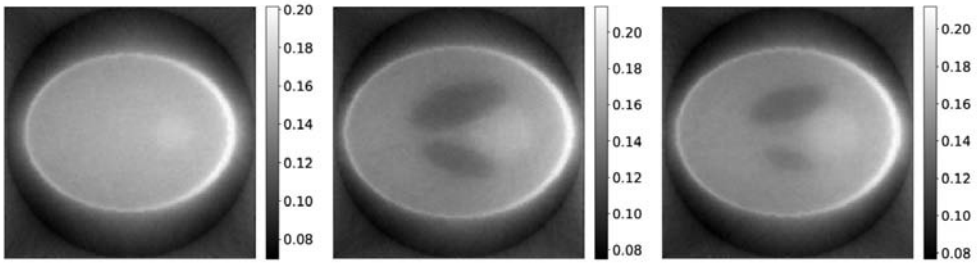


Figure 2.8: MLEM_1 reconstruction of test images

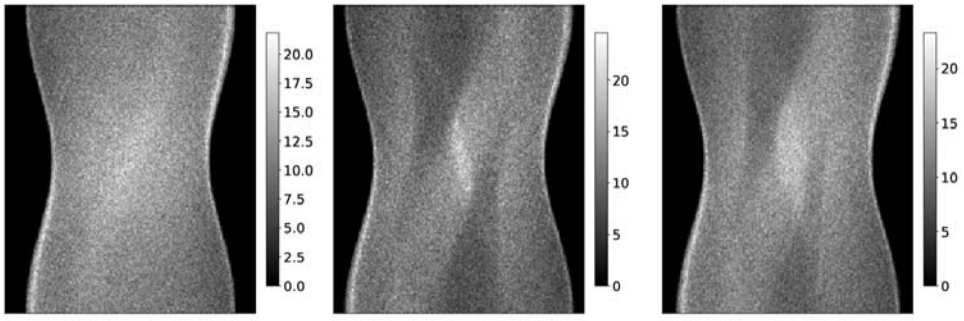


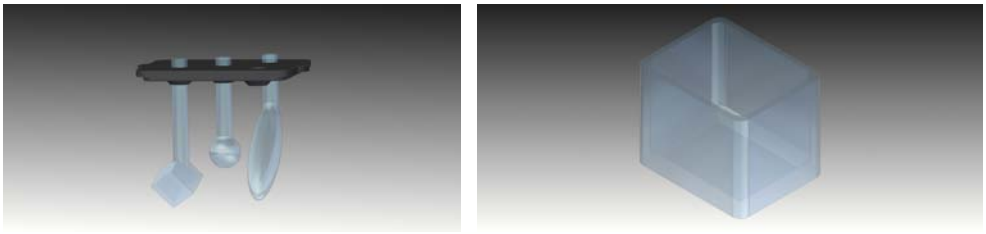
Figure 2.9: Noisy sinograms obtained from test images

2.3 miniPET Data

2.3.1 Train Phantoms

Three different phantoms have been designed in order to generate a miniPET training data set. Each phantom has a specific design concept thought to introduce relevant information in the training data. After the CAD design process each phantom is 3D printed and ready to be used for miniPET measurements.

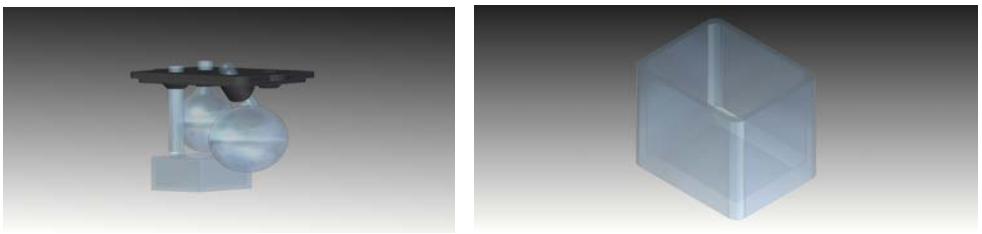
- **Train Phantom #1:** The purpose of this phantom is to introduce in the training set images representing small objects in a volume that will be quite empty. Three different shapes are present: an ellipsoid, a sphere and a cube. Figure 2.10 shows the CAD project of the first training phantom.



(a) Insert (b) Container

Figure 2.10: Train phantom #1

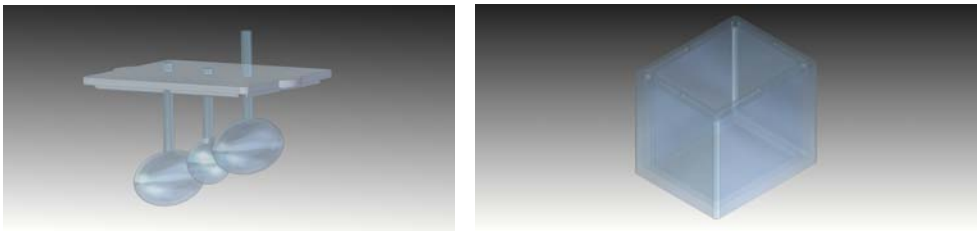
- **Train Phantom #2:** The purpose of this phantom is to introduce in the training set images representing big objects in a volume that will be quite full. Three different shapes are present: an ellipsoid, a sphere and a parallelepiped. Figure 2.11 shows the CAD project of the second training phantom.



(a) Insert (b) Container

Figure 2.11: Train phantom #2

- **Train Phantom #3:** The purpose of this phantom is to introduce in the training set images that are similar to those obtained from the Shepp-Logan phantom. Three different objects are present: two ellipsoids and a sphere, an inner wall is also raised in the container in order to obtain two separated background volumes. Figure 2.12 shows the CAD project of the third training phantom.

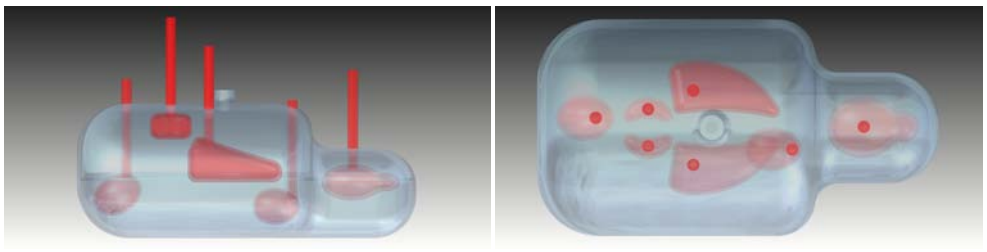


(a) Insert (b) Container

Figure 2.12: Train phantom #3

2.3.2 Test Phantom

A single test phantom has been designed in order to build the test data set. The design concept of this phantom is to obtain images that are similar to those obtained from a mouse PET measure. Only the main organs that are visible in a mouse PET image have been considered, namely the brain, the heart, the lungs, the kidneys and the bladder. All these organs have been placed into a mouse shaped shell in a position that is as close as possible to the real mouse anatomy. After the CAD design process the mouse phantom is 3D printed and ready to be used for miniPET measurements. Figure 2.13 shows two different views of the CAD project of the mouse phantom.



(a) Mouse Phantom side view (b) Mouse phantom top view

Figure 2.13: Mouse Phantom

2.3.3 miniPET Training Data Generation

In order to build the miniPET training data set six measurements have been performed for each train phantom. Considering the low number of measurements that could be performed high variability between different measurements to be made with the same phantom is the key concept that has been considered when designing a measure. In order to achieve this goal three different radioactivity concentration intervals have been set:

- **High**: Radioactivity concentrations in the interval $[1.5, 3]\frac{\text{Mbq}}{ml}$, to be used for the objects of the phantom,
- **Medium**: Radioactivity concentrations in the interval $[0.7, 1.5]\frac{\text{Mbq}}{ml}$, to be used for the objects of the phantom,
- **Low**: Radioactivity concentrations in the interval $[0.05, 0.5]\frac{\text{Mbq}}{ml}$, to be used for the background of the phantom.

Various radioactivity concentrations of fluorodeoxyglucose $[^{18}F]FDG$ belonging to the previously described intervals have been prepared and injected into the different objects of a phantom always trying to achieve high variability for five different measurements. The sixth measure is performed leaving all the objects empty and injecting radioactivity only in the background. The full list of performed measurements with the corresponding activity concentrations of each object and background can be found in the `Measurments Description.txt` file uploaded on my `GitHub`.

The different activity concentrations that will be injected in the phantom and its total radioactivity are simulated before performing a measure using an `Excel` spreadsheet. A different spreadsheet is developed for each phantom considering the differences in the volumes of the different objects of each phantom. All the spreadsheets have been developed to be robust to differences in the total available radioactivity to be diluted in the different volumes since we did not have complete control over this parameter. In the spreadsheet we also compute the different activity concentrations after an hour considering the radioactive decay of $[^{18}F]FDG$, this has proved to be very useful when designing multiple measurements to be performed one after the other. The three `Excel` spreadsheets `MeasureSimulatorPX.xlsx` can be found on my `GitHub`.

A measure consists of sixty acquisition shots of one minute each for an overall one hour acquisition window. Data obtained from each shot are corrected considering the $[^{18}F]FDG$ decaying time with the miniPET software.

The command used to start a miniPET acquisition has the following layout:

```
mpdaq -f18 -tprogr Folder measureName 1 60 -min -cli
```

After the acquisition, data are processed in order to obtain different levels of noise. Different levels of noise are obtained considering different acquisition windows that can be achieved with a single measure by using the cumulative sum operator on the one minute shots measurements. Starting from the sixty one minute shots we thus obtain sixty measurements with increasing acquisition times starting from one minute up to one hour, data coming from each measure are finally converted into direct sinograms that can be processed in the *Python* environment. All the computations needed to process the acquired data are carried out with the miniPET software, the commands used have the following layout:

```
lr5gen -sum -rshot 0 stop input.clm output.3dlor.lr5

lr5bin input.3dlor.lr5 output.2dlor.lr5

lr5bin -sin -mnc input.2dlor.lr5 output.sino.mnc
```

For each measure, the sixty direct sinograms obtained after the data processing step are coupled with the same target image when added to the training data set.

The target image is obtained using the miniPET reconstruction software that performs a twenty iteration MLEM reconstruction considering all data acquired with a one hour acquisition window, note that with the miniPET software it is possible to reconstruct an image using also indirect sinograms that can not be processed in the *Python* environment, the reconstruction result is thus better. Figure 2.14 shows a miniPET training data set couple.

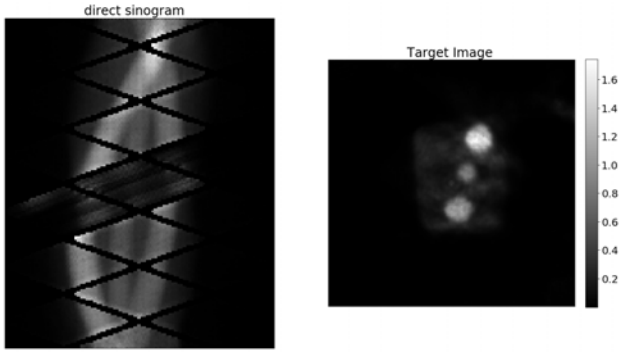


Figure 2.14: miniPET training data set couple

The command used to obtain the target image has the following layout:

```
rec -v5 -thread 8 input.2dlor.lr5 GT.mnc
```

All the commands needed to process the acquired data and obtain the target image from a measure are applied using a *Bash* file named `processing.sh` that can be found on my `GitHub`.

2.3.4 miniPET Test Data Generation

Two measurements have been performed with the mouse-like test phantom in order to build a test data set. The measurement protocol is the same used for the training measurements, described in the previous section. The activity concentration levels for the various objects of the phantom are chosen to be similar to those used in the objects of the training phantoms. The difference between the two test measurements lies in the mouse orientation, in the second measure the mouse is rotated $90°$. Similar activity concentration levels are used for the two test measurements instead.

2.4 Network Architectures

2.4.1 Denoising Architectures

Two architectures are considered for denoising problems. The first architecture is a U-Net which structure is depicted in Figure A.15, the second architecture is an encoder followed by a decoder and is obtained by removing the skip connections from the U-Net architecture as Figure A.16 shows. A detailed description of the components of these architectures can be found in Appendix A, section A.2.3. Architectures with three, four and five layers are considered, six layers architectures are not considered since after six poolings the resulting image has a size lower than the 3×3 convolutional kernel. The number of parameters of each considered network is reported in Table 2.1.

Table 2.1: Number of parameters of denoising networks

network Architecture	**# of layers**	**# of parameters**
U-Net	3	2,143,329
U-Net	4	8,636,769
U-Net	5	34,599,777
Encoder-Decoder	3	1,949,793
Encoder-Decoder	4	7,853,409
Encoder-Decoder	5	31,457,121

All considered architectures have been developed using the *Pytorch* libraries and tools of *Python* programming language.

2.4.2 Learned Update Reconstruction Architectures

The *Learned Update* (LU) reconstruction aims to improve results obtained with the simple denoising approach by iteratively update the reconstruction using a series of U-Nets and reintroducing the information carried by the original noisy data at each iteration.

The first iteration of the algorithm is similar to the denoising approach, the difference is that instead of using the MLEM_1 algorithm to obtain a first reconstruction to be given as input to the first U-Net Λ_{θ_0}, we use $\mathcal{R}(\cdot) = \frac{\mathcal{T}^*(\cdot)}{||\mathcal{T}||^2}$, that is, the result of the adjoint of the forward operator $\mathcal{T}^*(\cdot)$ normalized by its square norm $||\mathcal{T}^*||^2 = ||\mathcal{T}||^2$. The result is a more rough reconstruction,

with respect to MLEM_1, but the computation time is faster. The result of this first iteration is $x^{(0)} = \Lambda_{\theta_0}(\mathcal{R}(s))$ then the iterative approach starts. We first use the forward operator $\mathcal{T}(\cdot)$ to obtain the sinogram of the previous iteration reconstruction $s^{(i-1)} = \mathcal{T}(x^{(i-1)})$ and use the result to compute the difference with the original input data $s^{(i-1)} - s$. We then use $\mathcal{R}(\cdot)$ to reconstruct the image corresponding to this data difference $x_\Delta^{(i-1)} = \mathcal{R}(s^{(i-1)} - s)$, this image is then given as input to a U-Net Λ_{θ_i} together with the previous iteration reconstruction $x^{(i-1)}$. Finally the current iteration reconstruction is computed by adding the Λ_{θ_i} output to the previous iteration $x^{(i-1)}$. This process is repeated for N_{LU} iterations.

$$x^{(i)} = x^{(i-1)} + \Lambda_{\theta_i}(x^{(i-1)}, x_\Delta^{i-1}) \qquad i = 1, \cdots, N_{\text{LU}} - 1 \tag{2.3}$$

The full mathematical description of the LU algorithm is reported in Algorithm 1 and the corresponding block diagram is depicted in Figure 2.15.

The Λ_{θ_0} architecture is exactly the same as the one used in the denoising approach with three layers of depth, the Λ_{θ_i} with $i = 1, \cdots, N_{\text{LU}} - 1$ architecture is the same of Λ_{θ_0}, but without the ReLU on the last 1×1 convolution, that has been removed to allow a negative update of the previous iteration reconstruction, and with a two channel input layer instead of a single one.

The forward operator $\mathcal{T}(\cdot)$ and its adjoint $\mathcal{T}^*(\cdot)$ are obtained from the `STIR` library implemented in *Python* programming language.

Algorithm 1 Learned Update Reconstruction

Given: s, N_{LU}
Using: $\mathcal{R}(\cdot) = \frac{\mathcal{T}^*(\cdot)}{||\mathcal{T}||^2}$
$x^{(0)} = \Lambda_{\theta_0}(\mathcal{R}(s))$
for $i = 1, \cdots, N_{\text{LU}} - 1$ **do**
    $s^{(i-1)} = \mathcal{T}(x^{(i-1)})$
    $x_\Delta^{(i-1)} = \mathcal{R}(s^{(i-1)} - s)$
    $x^{(i)} = x^{(i-1)} + \Lambda_{\theta_i}(x^{(i-1)}, x_\Delta^{(i-1)})$
end for
return $x^{(i)}$

Two, three and four iterations of the LU algorithm are considered, one iteration is just denoising with a slightly different input image meanwhile five or more iterations are computationally heavy and not meaningful as explained in Chapter 4, section 4.4.

A *Learned Update with Memory* (LUM) architecture is also considered. This approach is slightly different from the previous one, the difference be-

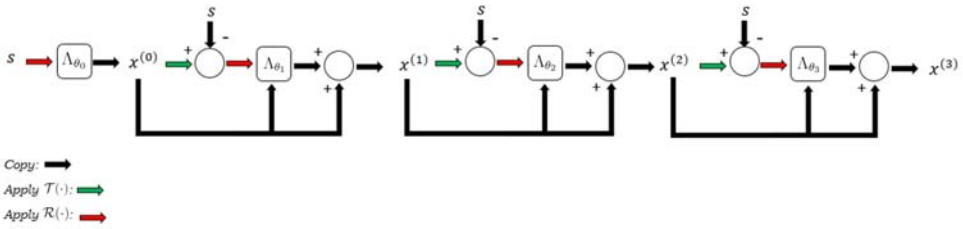


Figure 2.15: Learned update architecture, four iterations

ing that to each U-Net Λ_{θ_i} with $i = 1, \cdots, N_{\text{LU}} - 1$ we give as input also all the previous iterations reconstructions, $x^{(0)}, \cdots, x^{(i-1)}$. By doing this we allow the network to consider also all the previous iterations reconstructions when computing the update to be applied to the current one. The mathematical description of the LUM algorithm is reported in Algorithm 2 and the corresponding block diagram is depicted in Figure 2.16.

Algorithm 2 Learned Update with Memory Reconstruction

Given: s, N_{LU}
Using: $\mathcal{R}(\cdot) = \frac{\mathcal{T}^*(\cdot)}{||\mathcal{T}||^2}$
$x^{(0)} = \Lambda_{\theta_0}(\mathcal{R}(s))$
for $i = 1, \cdots, N_{\text{LU}} - 1$ **do**
    $s^{(i-1)} = \mathcal{T}(x^{(i-1)})$
    $x_{\Delta}^{(i-1)} = \mathcal{R}(s^{(i-1)} - s)$
    $x^{(i)} = x^{(i-1)} + \Lambda_{\theta_i}(x^{(0)}, \cdots, x^{(i-1)}, x_{\Delta}^{(i-1)})$
end for
return $x^{(i)}$

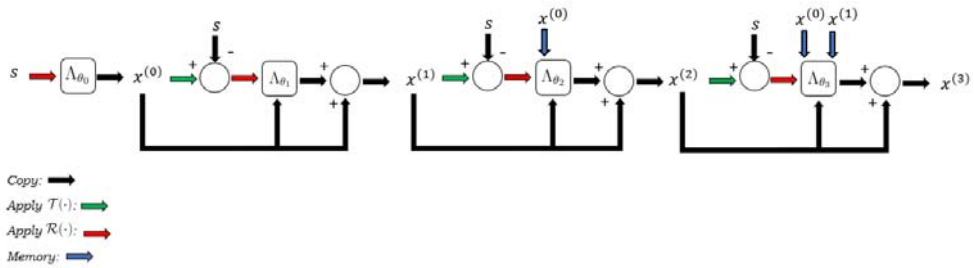


Figure 2.16: Learned update with memory architecture, four iterations

For the LUM algorithm only four iterations are considered since two iterations lead to the same architecture of the LU algorithm, three iterations have

only one memory element that does not significantly affect the final result and five or more iterations are computationally heavy. The total number of parameters of both LU and LUM considered architectures are reported in Table 2.2.

All considered architectures have been developed using the *Pytorch* libraries and tools of *Python* programming language.

Table 2.2: Number of parameters of LU and LUM architectures

Algorithm	**# of iterations**	**# of parameters**
LU	2	4,286,946
LU	3	6,430,563
LU	4	8,574,180
LUM	4	8,575,044

2.4.3 Learned Primal-Dual Reconstruction Architectures

The *Learned Primal-Dual Reconstruction* (LPD) aims to improve results obtained with the LU algorithm by applying the iterative update also in the data space and sharing information between data space and image space at each iteration.

Each iteration consists of two U-networks applied one after the other. The first iteration is simpler and thus different than all the others. The first U-Net Ξ_{θ_0} works in the data domain and has the noisy sinogram s as input, its output $h^{(0)} = \Xi_{\theta_0}(s)$ is then converted into an image using $\mathcal{R}(\cdot) = \frac{\mathcal{T}^*(\cdot)}{||\mathcal{T}||^2}$. The result of this operation $h^{(0)}_{\text{img}} = \mathcal{R}(h^{(0)})$ is then given as input to the second U-Net Λ_{θ_0} that works in the image domain, the output of this network $f^{(0)} = \Lambda_{\theta_0}(h^{(0)}_{\text{img}})$ is the first iteration reconstruction of the algorithm. A generic iteration $i > 0$ of the LPD algorithm starts with a U-Net Ξ_{θ_i} that works on the data space and recevies as inputs the noisy sinogram s, the outputs of all previous Ξ_θ networks $h^{(0)}, \cdots, h^{(i-1)}$ and the sinogram obtained by applying the forward operator $\mathcal{T}(\cdot)$ to the previous iteration reconstruction $f^{(i-1)}_{\text{data}} = \mathcal{T}(f^{(i-1)})$. In order to obtain $h^{(i)}$, that is the updated version of $h^{(i-1)}$, the output of Ξ_{θ_i} is added to $h^{(i-1)}$:

$$h^{(i)} = h^{(i-1)} + \Xi_{\theta_i}(s, h^{(0)}, \cdots, h^{(i-1)}, f^{(i-1)}_{\text{data}}) \qquad i = 1, \cdots, N_{\text{LPD}} - 1 \quad (2.4)$$

After this computation, a U-Net Λ_{θ_i} that works on the image space is used to update the previous iteration reconstruction $f^{(i-1)}$. The inputs of Λ_{θ_i} are all the previous iterations reconstructions $f^{(0)}, \cdots, f^{(i-1)}$ and the image obtained by applying $\mathcal{R}(\cdot)$ to $h^{(i)}$, $h^{(i)}_{\text{img}} = \mathcal{R}(h^{(i)})$. The current iteration reconstruction $f^{(i)}$ is obtained by summing the Λ_{θ_i} network output to the previous iteration reconstruction $f^{(i-1)}$:

$$f^{(i)} = f^{(i-1)} + \Lambda_{\theta_i}(f^{(0)}, \cdots, f^{(i-1)}, h^{(i)}_{\text{img}}) \qquad i = 1, \cdots, N_{\text{LPD}} - 1 \quad (2.5)$$

This process is repeated for N_{LPD} iterations.

The full mathematical description of the LPD algorithm is reported in Algorithm 3 and the corresponding block diagram is depicted in Figure 3.7.

Algorithm 3 Learned Primal-Dual Reconstruction

Given: s, N_{LPD}
Using: $\mathcal{R}(\cdot) = \frac{\mathcal{T}^*(\cdot)}{||\mathcal{T}||^2}$
$h^{(0)} = \Xi_{\theta_0}(s)$
$h^{(0)}_{\text{img}} = \mathcal{R}(h^{(0)})$
$f^{(0)} = \Lambda_{\theta_0}(h^{(0)}_{\text{img}})$
for $i = 1, \cdots, N_{\text{LPD}} - 1$ **do**
    $f^{(i-1)}_{\text{data}} = \mathcal{T}(f^{(i-1)})$
    $h^{(i)} = h^{(i-1)} + \Xi_{\theta_i}(s, h^{(0)}, \cdots, h^{(i-1)}, f^{(i-1)}_{\text{data}})$
    $h^{(i)}_{\text{img}} = \mathcal{R}(h^{(i)})$
    $f^{(i)} = f^{(i-1)} + \Lambda_{\theta_i}(f^{(0)}, \cdots, f^{(i-1)}, h^{(i)}_{\text{img}})$
end for
return $f^{(i)}$

The Λ_{θ_i} with $i = 0, \cdots, N_{\text{LU}} - 1$ architectures are the same as the one used in the denoising approach with three layers of depth but without the ReLU on the last 1×1 convolution, that has been removed to allow a negative update of the previous iteration reconstruction, and with a multiple channel input layer instead of a single one. The Ξ_{θ_i} architectures are the same as the Λ_{θ_i} but paddings have been adjusted to fit the sinogram size that is 180×147 instead of 147×147.

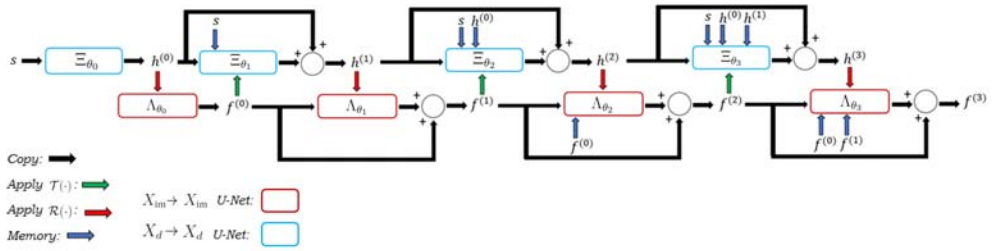


Figure 2.17: Learned Primal-Dual architecture, four iterations

The forward operator $\mathcal{T}(\cdot)$ and its adjoint $\mathcal{T}^*(\cdot)$ are obtained from the `STIR` library implemented in *Python* programming language.

One, two, three and four iterations of the LPD algorithm are considered, five or more iterations are computationally too heavy.

The total number of parameters of the considered LPD architectures are reported in Table 2.3.

Table 2.3: Number of parameters of LPD architectures

Algorithm	**# of iterations**	**# of parameters**
LPD	1	4,286,658
LPD	2	8,574,180
LPD	3	12,862,278
LPD	4	17,150,952

All considered architectures have been developed using the *Pytorch* libraries and tools of *Python* programming language.

2.5 Training procedures

2.5.1 Denoising Networks Training

Denoising networks are trained using a checkpoint strategy to reduce the computational time and allow to train multiple networks at the same time. A checkpoint consists of a train with 25000 train images for 25 epochs with batch size equal to 20 and learning rate equal to $1.5e^{-3}$ using `torch.optim.Adam` as optimizer and the Smooth L^1 loss, `nn.SmoothL1Loss()`. At the end of a training checkpoint the model parameters and the status of the optimizer are saved in a `.tar` file, at the start of the next checkpoint this file is loaded and training will resume from where it was previously stopped. The maximum number of checkpoints has been set to four, equivalent to a train with 100000 images for 100 epochs. This maximum number has been fixed mainly considering the computational time needed since a training checkpoint requires from four to six hours to be completed, depending on the depth of the network, and that a total of six networks need to be trained.

The *Python* codes to train the two denoising architectures can be found on my `GitHub`.

2.5.2 Learned Update Training

The networks in the Learned Update algorithm are trained using a *Progressive Learning* (PL) strategy to reduce the computational time and allow the architecture to work as intended. The PL training starts with two networks, corresponding to two iterations of the algorithm, the initialization of the parameters of the first network Λ_{θ_0} is performed by loading the parameters obtained from the training of the three layers denoising U-Net with three checkpoints and a variable level of noise. The second network Λ_{θ_1} is initialized according to the *Xavier* initialization [1]. The obtained LU architecture is then trained with 100000 training data for 5 epochs with batch size equal to 10 and learning rate equal to $1.5e^{-3}$ using `torch.optim.Adam` as optimizer and the Smooth L^1 loss, `nn.SmoothL1Loss()`. In order to train architectures with $N_{\text{LU}} > 2$ iterations following the PL strategy we add only one network, thus increasing the number of iterations by one, initialize it with the *Xavier* initialization and initialize all the other networks with the parameters obtained training a LU architecture with $N_{\text{LU}} - 1$ iterations. Then training is performed with 125000 train couples with a variable level of noise for 3 epochs with batch size equal to 10 and learning rate equal to $1.5e^{-3}$ using `torch.optim.Adam` as op-

timizer and the Smooth L^1 loss, `nn.SmoothL1Loss()`. This process is repeated until the number of iterations is equal to the desired one.

The maximum number of training data, epochs and iterations has been fixed mainly considering the computational time needed since the forward operator $\mathcal{T}(\cdot)$ and the reconstruction $\mathcal{R}(\cdot)$ are slow and must be used more times when the number of iterations increases. The training of the four iterations LU architecture is completed after a week of training considering the PL strategy but it gives results also for the two and three iterations of the algorithm.

A standard training approach has also been used in order to see whether the PL strategy is strictly necessary to obtain a good training. Following the standard training approach all the desired N_{LU} networks are initialized with the *Xavier* initialization and trained with 125000 train data with a variable level of noise for 3 epochs with batch size equal to 10 and learning rate equal to $1.5e^{-3}$ using `torch.optim.Adam` as optimizer and the Smooth L^1 loss, `nn.SmoothL1Loss()`.

The *Python* code to train the LU architecture can be found on my `GitHub`.

2.5.3 Learned Update With Memory Training

The four networks of the learned update with memory algorithm are trained with 200000 train data with a variable level of noise for 3 epochs with batch size equal to 10 and learning rate equal to $1.5e^{-3}$ using `torch.optim.Adam` as optimizer and the Smooth L^1 loss, `nn.SmoothL1Loss()`. The model parameters are initialized by loading the four iterations learned update algorithm parameters for all the layers of the various networks except the first layer of networks Λ_{θ_2} and Λ_{θ_3} that have a different number of input channels, these layers are initialized with the *Xavier* initialization.

The *Python* code to train the LUM architecture can be found on my `GitHub`.

2.5.4 Learned Primal-Dual Training

The networks in the Learned Primal-Dual algorithm are trained using a mixed strategy that combines the Progressive Learning and the checkpoints strategies. An extra training step is needed at the start, the first U-Net of the algorithm Ξ_{θ_0} is initially trained alone with 100000 train couples where the network input is the noisy sinogram with a variable level of noise and the ground truth is the noise free sinogram, this step is needed so that the U-Net will learn that its purpose is to denoise a sinogram. The training is 1 epoch long with batch size equal to 5 and learning rate equal to $1.5e^{-3}$ using

`torch.optim.Adam` as optimizer and the Smooth L^1 loss. The second U-Net Λ_{θ_0} is then added to the algorithm so as to produce one iteration of the LPD algorithm. Now the proper training starts following the PL strategy, the parameters of the Ξ_{θ_0} network are initialized by loading the parameters obtained with the previously described training for this network and the parameters of the Λ_{θ_0} network are initialized by loading the parameters obtained from the training of the three layers denoising U-Net with three checkpoints and a variable level of noise. The one iteration LPD architecture is then trained with 125000 training data with a variable level of noise for 3 epochs with batch size equal to 5 and learning rate equal to $1.5e^{-3}$ using `torch.optim.Adam` as optimizer and the Smooth L^1 loss. When a new iteration is added, $N_{\text{LPD}} > 1$, two new U-Net Ξ_{θ_i} and Λ_{θ_i} are considered in the architecture. All the previous iterations networks parameters are initialized by loading the parameters obtained with the training of an architecture with $N_{\text{LPD}} = i - 1$, according to the PL training strategy. The current iteration networks parameters are initialized by loading the parameters of the last two networks of a previously trained architecture with $N_{\text{LPD}} = i - 1$. Finally the parameters of the first layer of all the networks are initialized with the *Xavier* initialization in order to add some randomness to the initialization. The LPD architecture is then trained with 125000 training data with a variable level of noise for 3 epochs with batch size equal to 5 and learning rate equal to $1.5e^{-3}$ using `torch.optim.Adam` as optimizer and the Smooth L^1 loss, `nn.SmoothL1Loss()`. This process is repeated until the number of iterations is equal to the desired one. After the PL strategy further training may be performed using the checkpoint strategy for architectures with a fixed number of iterations in order to obtain a better convergence. The parameters of the architecture are initialized by loading the parameters obtained with the PL strategy and then multiple checkpoints can be used to continue the training while keeping fixed the number of iterations. A checkpoint consists of a train with 200000 train data with a variable level of noise for 3 epochs with batch size equal to 15 and learning rate equal to $1.5e^{-3}$ using `torch.optim.Adam` as optimizer and the Smooth L^1 loss, `nn.SmoothL1Loss()`. At the end of a training checkpoint the model parameters and the status of the optimizer are saved in a `.tar` file, at the start of the next checkpoint this file is loaded and training will resume from where it was previously stopped. The training process is continued until there is little difference between results obtained with the previous and current checkpoint.

The training of the three iterations LPD architecture using the PL strategy is completed after a week, then each checkpoint is completed after two days.

The *Python* code to train the LPD architecture can be found on my `GitHub`.

2.6 miniPET Data Training

Two different approaches have been developed in order to train a complex algorithm with few miniPET data. The parameters of the model are firstly initialized with the parameters obtained by training the networks only with synthetic data following the previously presented approaches, then the miniPET data training can start according to the following strategies:

- **miniPET only training**: Only miniPET data are inserted in the training set, the training set size is thus 35700 pairs, since we created 60 different noise levels for each measurement and from each 3D volume we can extract 35 two dimensional slices,
- **Hybrid training**: In this case a mix of miniPET data and synthetic data are inserted in the training set. The number of synthetic data is fixed equal to a quarter of the miniPET data set size. The total hybrid training set size is thus 44625.

The training is 10 epochs long with batch size equal to 15 and learning rate equal to $1.5e^{-3}$ using `torch.optim.Adam` as optimizer and the Smooth L^1 loss, `nn.SmoothL1Loss()` for both approaches. After each epoch the performance of the trained architecture are evaluated on the miniPET test data set using the Smooth L^1 loss and the parameters of the model are saved. The model parameters that lead to the smallest loss on the miniPET test data set are chosen as the final ones.

The training of the three iterations LPD architecture using the Hybrid training approach is completed after 36 hours, 30 hours are needed to train the same architecture using the miniPET only approach.

The *Python* codes to extend the training of the LPD architecture with miniPET data can be found on my `GitHub`.

2.7 Performance Evaluation

Results are evaluated considering three factors in the following order:

1. **Visual inspection**: This is a qualitative way to determine whether a result is good or not; absence of artifacts, good shapes reconstruction, correct dynamic range and correct region intensities are evaluated,

2. **PSNR**: Index that evaluates the level of noise of an image, both the obtained value for the denoised image and the increment in PSNR with respect to the MLEM_{10} reconstruction (ΔPSNR) are considered,

3. **SSIM**: Index that evaluates the structural similarity, both obtained value for the denoised image and the increment in SSIM with respect to the MLEM_{10} reconstruction (ΔSSIM) are considered.

The visual inspection is considered before all the others since it is more important to have results without artifacts and with correctly represented shapes instead of having bad shapes, lots of artifact but low noise. If the result is bad from a visual inspection PSNR and SSIM are thus not considered.

Chapter 3

Results

In this chapter results of both denoising and reconstruction problems are presented. We start with results obtained for denoising problems with different architectures on the synthetic test images set, then results obtained for reconstruction problems with different algorithms are presented always on the synthetic test images set, finally LPD results on the miniPET test phantom are presented.

3.1 Denoising Results

3.1.1 Encoder-Decoder Test Images Set Performance

All the following results have been obtained by training the networks with the checkpoint procedure described in Chapter 2, section 2.4.1 and a fixed noise level. Test images have been generated with a fixed noise level as well.

The three layers Encoder-Decoder leads to a good result on the test set images when trained with three checkpoints.

Figure 3.1 shows the ground truth, the MLEM_{10} reconstruction, the network input MLEM_1 reconstruction and the three layers Encoder-Decoder denoiser output of a test image. The denoised image has a better quality than the MLEM_{10} reconstruction.

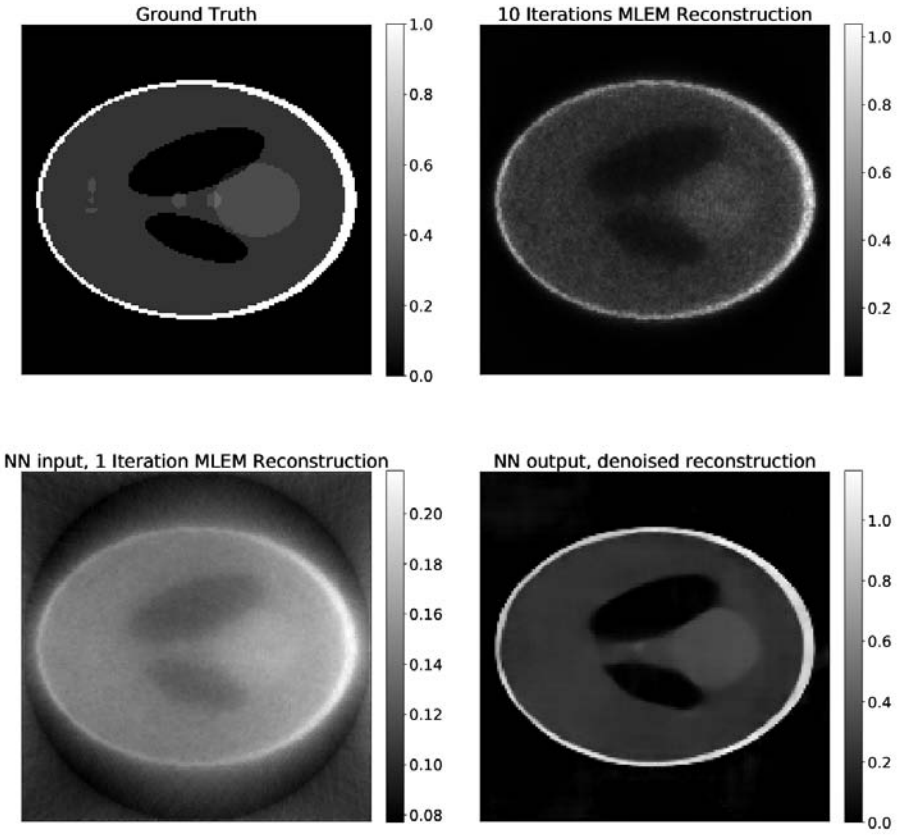


Figure 3.1: Encoder-Decoder 3 layers: ground truth, MLEM_{10}, CNN input, CNN output

3.1.2 U-Net Test Images Set Performance

All the following results have been obtained by training the networks with the checkpoint procedure described in Chapter 2, section 2.4.1 and a fixed noise level. Test images have been generated with a fixed noise level as well.

The three layers U-Net leads to a good result when trained with four checkpoints. Results obtained with this denoiser are really similar to those obtained with the same depth Encoder-Decoder but with the U-Net architecture the original shape of both low contrast and high contrast objects is better preserved.

Figure 3.2 shows the ground truth, the MLEM_{10} reconstruction, the network input MLEM_1 reconstruction and the three layers U-Net denoiser output of a test image. The denoised image has a better quality than the MLEM_{10} reconstruction.

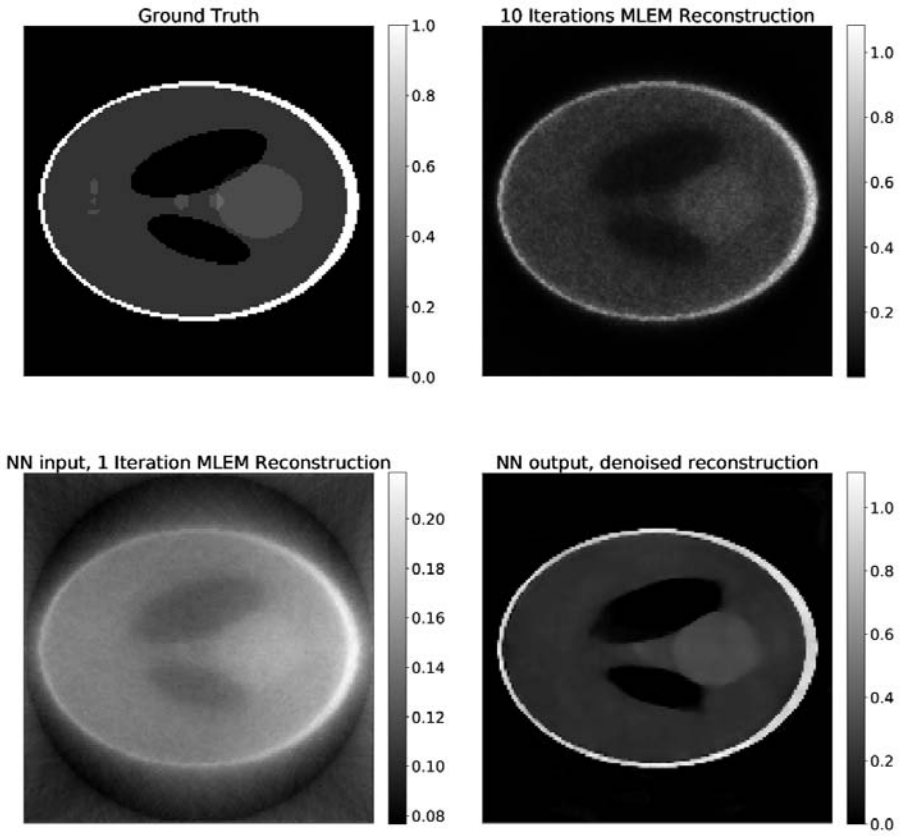


Figure 3.2: U-Net 3 layers: ground truth, MLEM_{10}, CNN input, CNN output

3.1.3 Encoder-Decoder vs U-Net Test Images Set

Results obtained with a three layers Encoder-Decoder and a three layers U-Net are similar from a visual inspection, in order to choose which architecture performs better for denoising problems we perform a quantitative analysis with figures of merit. The result of this analysis are reported in Table 3.1.

Considering together the visual inspection and the figures of merit the U-Net with three layers of depth is the best performing architecture on the test images set for denoising problems.

The *Python* codes to evaluate the results of the two denoising architectures can be found on my `GitHub`.

Table 3.1: Figures of Merit, U-Net vs Encoder-Decoder

	U-Net	Encoder-Decoder
PSNR	25.84	23.53
ΔPSNR	+5.75	+3.51
SSIM	0.934	0.833
ΔSSIM	+0.256	+0.156

3.2 Learned Update Results

3.2.1 Learned Update Test Images Set Performance

All the following results have been obtained by training the networks with the Progressive Learning procedure described in Chapter 2, section 2.4.2. Test images have been generated with a variable noise level.

Four iterations of the learned update algorithm lead to a good result, sometimes artifacts may appear near the terminal part of the high contrast ellipses as Figure 3.3 shows.

Three iterations of the learned update algorithm lead to the best result, both high and low contrast objects are very well reconstructed and the artifacts observed in the four iterations result are not present as Figure 3.4 shows. The LU reconstruction has a better quality than the MLEM_{10} reconstruction.

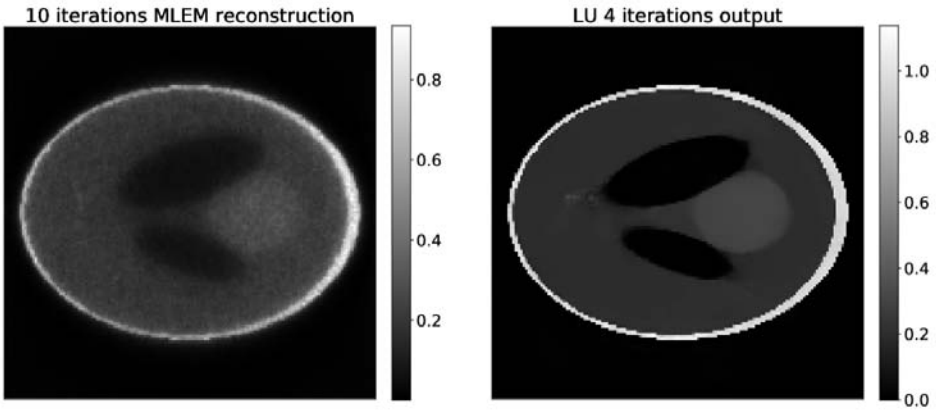


Figure 3.3: MLEM_{10} and LU 4 iterations output

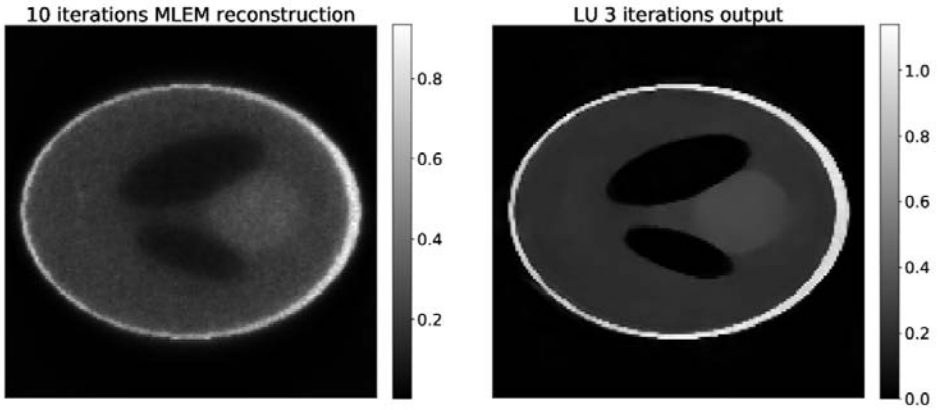


Figure 3.4: MLEM_{10} and LU 3 iterations output

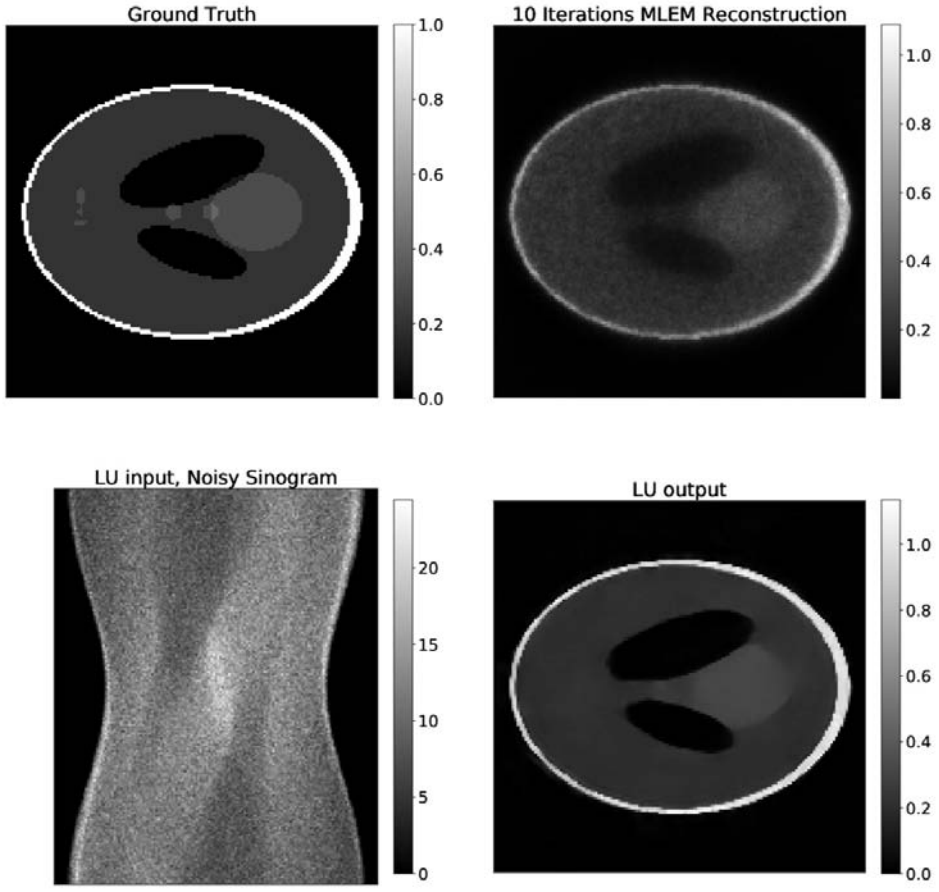


Figure 3.5: LU 3 iterations: ground truth, MLEM_{10}, LU input, LU output

Table 3.2: Figures of Merit, LU 3 iterations

	LU 3 Iterations
PSNR	25.94
ΔPSNR	+5.57
SSIM	0.89
ΔSSIM	+0.18

Figures of merit values for the three iterations learned update algorithm are reported in Table 3.2.

The *Python* code to evaluate the performance of the LU architecture can be found on my `GitHub`.

3.2.2 Learned Update with Memory Test Images Set Performance

The following results have been obtained by training the networks with the procedure described in Chapter 2, section 2.4.3. Test images have been generated with a variable noise level. The use of memory for the learned update algorithm does not improve the results, more artifacts are present near the terminal part of high contrast objects as Figure 3.6 shows.

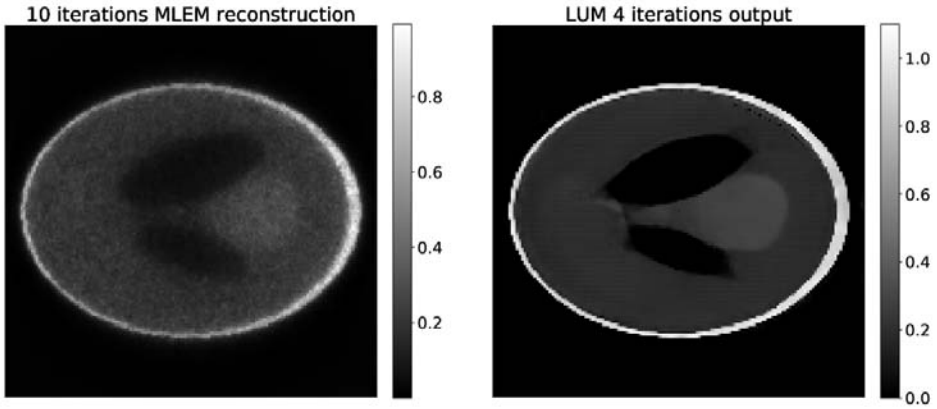


Figure 3.6: MLEM_{10} and LUM 4 iterations output

The *Python* code to evaluate the performance of the LUM architecture can be found on my `GitHub`.

3.3 Learned Primal-Dual Results

3.3.1 Learned Primal-Dual Test Images Set Performance

The following results have been obtained by training the networks with the procedure described in Chapter 2, section 2.4.4 Test images have been generated with a variable noise level. The best performing LPD architecture on the test images set is the the three iterations one. Both low and high contrast objects are well reconstructed and the dynamic range is almost identical to the ground truth one as Figure 3.8 shows. The LPD reconstruction has a better quality than the MLEM_{10} reconstruction as Figure 3.7 shows.

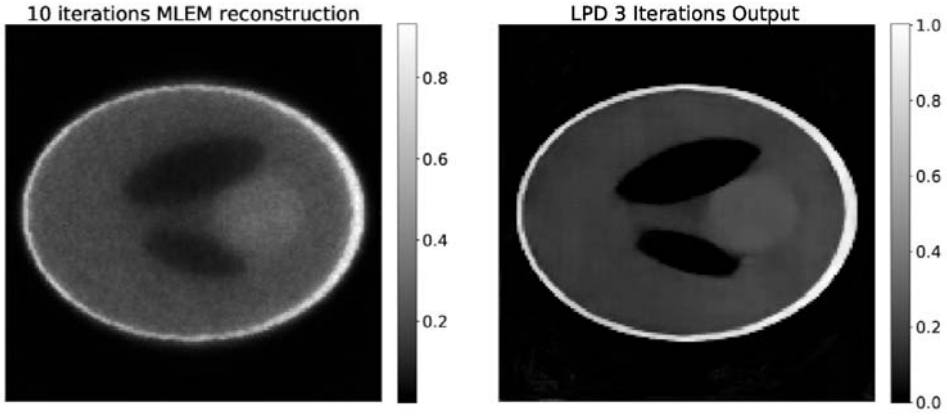


Figure 3.7: MLEM_{10} and LPD 3 iterations output

Figures of merit values for the three iterations learned primal-dual algorithm are reported in Table 3.3.

The *Python* code to evaluate the performance of the LPD architecture can be found on my `GitHub`.

Table 3.3: Figures of Merit, LPD 3 iterations

	LPD 3 Iterations
PSNR	24.36
ΔPSNR	+3.98
SSIM	0.87
ΔSSIM	+0.17

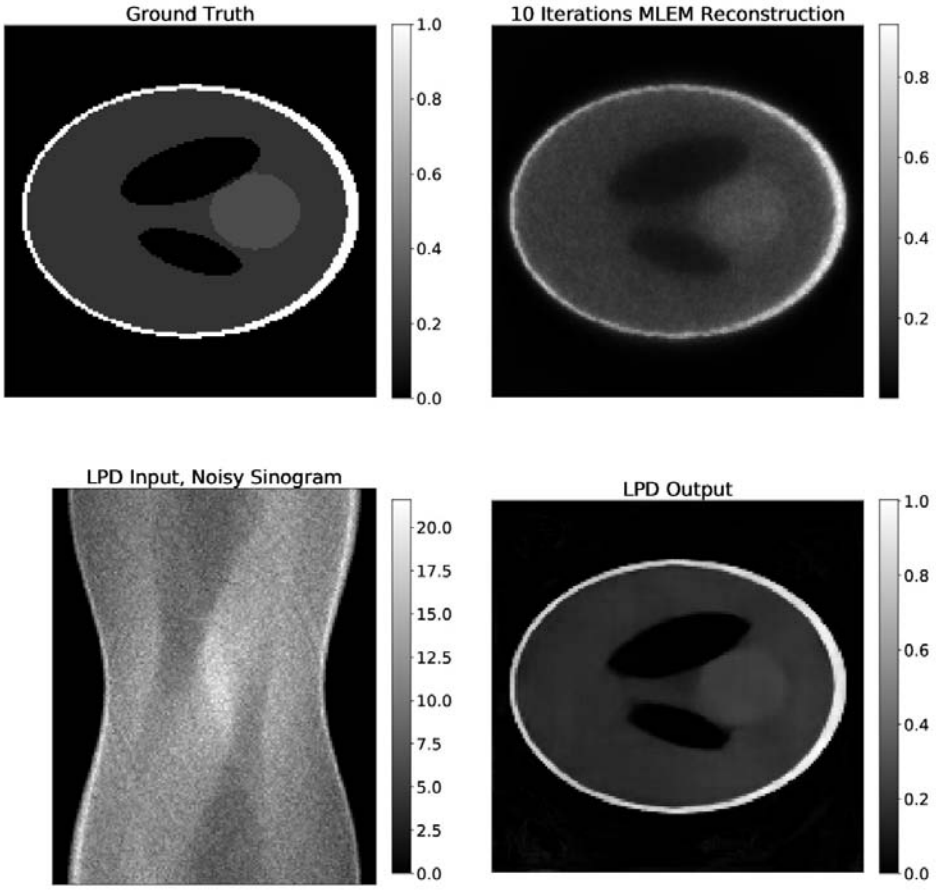


Figure 3.8: LPD 3 iterations: ground truth, MLEM_{10}, LPD input, LPD output

3.4 miniPET Data Reconstruction Results

The best results on miniPET data have been obtained by training the networks with the hybrid procedure described in Chapter 2, section 2.6.

Figure 3.9 shows the target image obtained with the miniPET reconstruction software using all measured data of an hour long acquisition, the MLEM_{20} reconstruction performed using only direct sinograms of a 51 minutes long acquisition and the three iterations LPD reconstruction performed using only direct sinograms of a 51 minutes long acquisition.

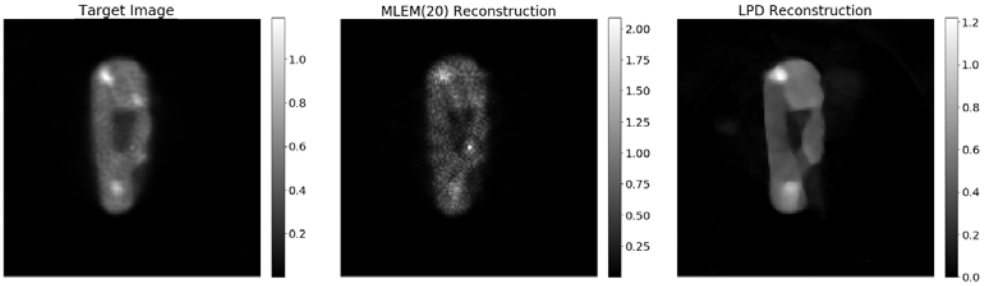


Figure 3.9: LPD 3 iterations: ground truth, MLEM_{20}, LPD output

The LPD reconstruction is better than the MLEM_{20} one when applied on the same data and is closer to the target reconstruction performed with more data and a nine minutes longer acquisition window. Note how the objects are more uniform and the dynamic range is closer to the target one for the LPD reconstruction when compared against the MLEM_{20} one.

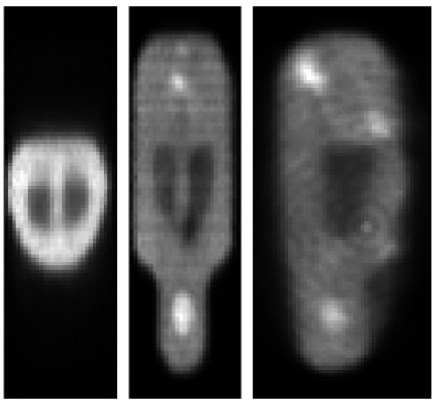
Figures of merit values for the three iterations learned primal-dual algorithm applied on miniPET data are reported in Table 3.4. The *Mean Squared Error* (MSE) is considered instead of the SSIM for miniPET data since SSIM does not give reliable results in this context. The decrease with respect to the MLEM_{20} MSE, ΔMSE, is reported as well.

Figure 3.10 show the target image, LPD and MLEM_{20} tomographic reconstruction. The LPD reconstruction performs better also when considering all three views of the mouse test phantom. Note how the stripes artifact due to the less data considered that can be observed in figure 3.10b is not present in the LPD reconstruction on the same data, figure 3.10c.

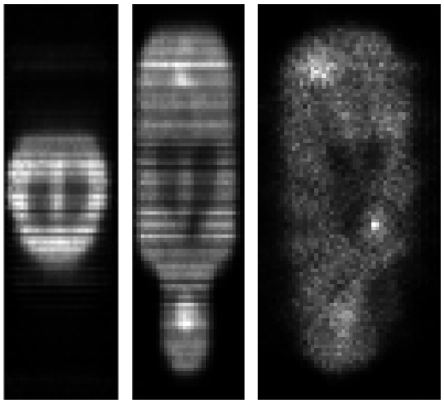
The *Python* code to evaluate the performance of the LPD architecture on miniPET data can be found on my `GitHub`.

Table 3.4: Figures of Merit, LPD 3 iterations miniPET data

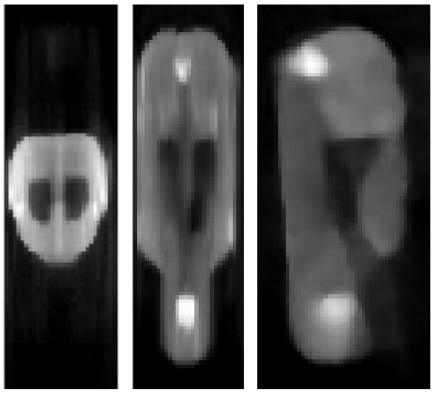
	LPD 3 Iterations miniPET
PSNR	25.20
ΔPSNR	+2.59
SSIM	0.00087
ΔSSIM	-0.00019



(a) Target image



(b) $MLEM_{20}$ reconstruction



(c) LPD reconstruction

Figure 3.10: Mouse test phantom: Target, LPD and $MLEM_{20}$ tomographic reconstructions

Chapter 4

Discussion

In this chapter the different approaches, problems and solutions that lead to the results using the described methods are discussed in detail. We start with denoising problems, then continue with reconstructions approaches and finally conclude with the miniPET data training extension problem.

4.1 On the Best Denoising Architecture

Figure 4.1 shows a denoising result obtained using an Encoder-Decoder with five layers of depth, this architecture does not perform well even with four training checkpoints. This is due to the model complexity that is too high for the task since we tried to train more the network, increasing the number of training checkpoints, but the results were similar to those obtained with four training checkpoints.

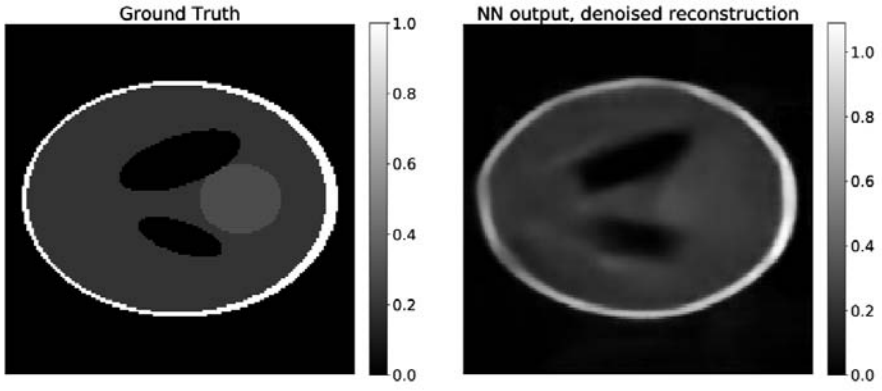


Figure 4.1: Encoder-Decoder 5 layers result, 4 training checkpoints

Figure 4.2 shows a denoising result obtained using a U-Net with five layers of depth, this architecture does not perform well even with a four checkpoints training. High contrast ellipses are well reconstructed but the dynamic range is quite different, low contrast objects are not well reconstructed despite being big and a grid like artifact covers the whole image. If compared to the five layers Encoder-Decoder the five layers U-Net is able to learn the task despite being really complex, this is due to the skip-connections that help the reconstruction in the up-convolution path of the U-Net thus making the learning process easier.

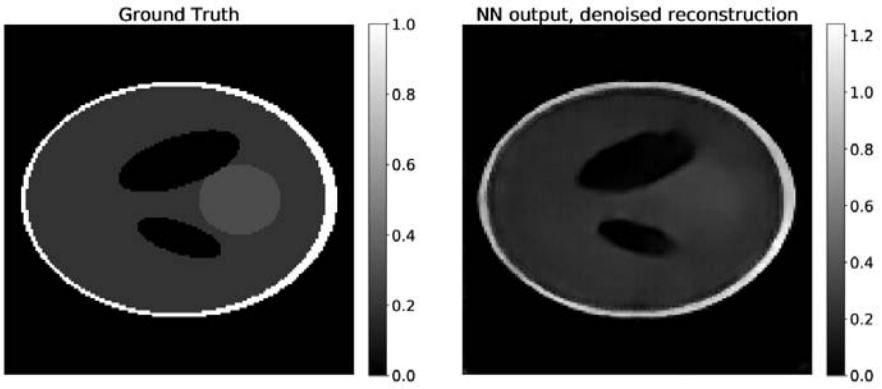


Figure 4.2: U-Net 5 layers result, 4 training checkpoints

The denoisning result obtained with a four layers Encoder-Decoder trained for four checkpoints is shown in Figure 4.3, this architecture performs better than the corresponding five layers one but the result is still not satisfactory, since many artifacts near the terminal part of the dark ellipses are present. Considering that training data are quite different from the test ones this model does not perform well because it is not general enough to handle this difference.

The four layers U-Net trained with three checkpoints performs better than the corresponding five layers one but the result is still not good enough. Low contrast objects are not well reconstructed and tend to be shadowed near the border as Figure 4.4 shows. The best training checkpoint for this architecture is the third one, more trained networks have more shape-related artifacts and less trained networks lead to a blurry result, as for the four layers Encoder-Decoder this is due to the model complexity that is too high and not general enough to handle the difference between training and test data.

The best number of layers of depth for denoisning architectures is three since the best results are obtained with such depth for both U-Net and Encoder-

Decoder architectures as discussed in Chapter 3, section 3.1. Models with this depth are complex enough to learn the task but also general enough to handle the difference between training and test data.

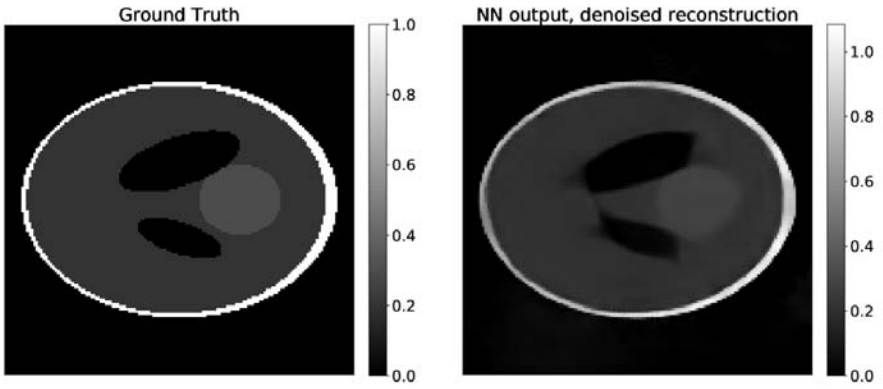


Figure 4.3: Encoder-Decoder 4 layers result, 4 training checkpoints

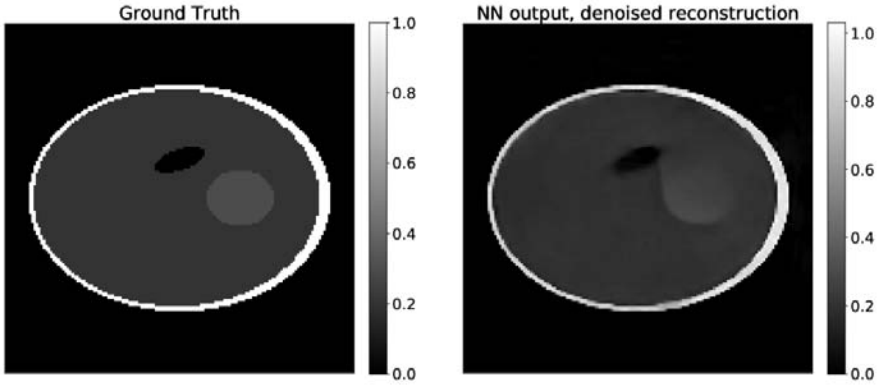


Figure 4.4: U-Net 4 layers result, 3 training checkpoint

4.2 Limits of Denoising approaches

Both the three layers deep networks are able to reconstruct very small low contrast objects if the random Poisson distributed noise level is not too high. If the particular realization of noise applied to the data is too high the small low contrast objects are not different enough from the surrounding space in the MLEM_1 reconstruction that is given as input to the network and the network is thus obviously not able to identify it.

Figure 4.5 shows a test image ground truth, the input given to a three layers Encoder-Decoder and the corresponding output. As we can see in the network input image there is a signal coming from the small low contrast object in the middle of the phantom and the network is thus able to represent it in the output.

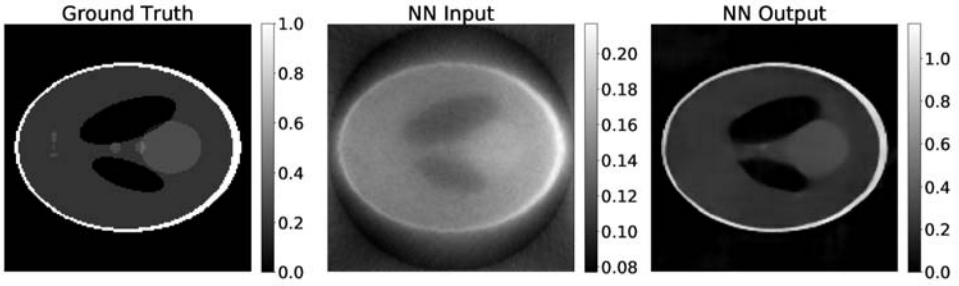


Figure 4.5: Encoder-Decoder 3 layers: ground truth, input and output

Figure 4.6 shows the same test image but with a different realization of noise applied to the data, now in the MLEM_1 reconstruction there is no signal coming from the small low contrast object in the center and the network is not able to represent it in the output. Note that this time there is a very low signal coming from the middle-right object in the phantom that is identified by the network and can be seen in the output image.

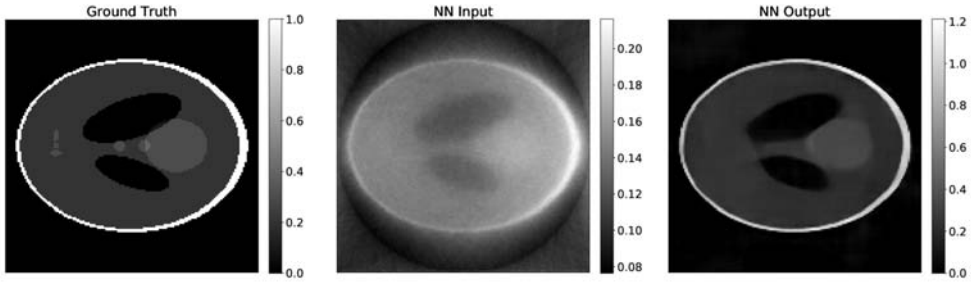


Figure 4.6: Encoder-Decoder 3 layers: ground truth, input and output

We can thus conclude that if the signal coming from an object is too low in the MLEM_1 that is given as input to the network this signal will be considered as noise from the network and will be removed from the output. This is the main limitation of denoising algorithms, they are highly dependent on the initial reconstruction that is performed to obtain the input image to be denoised, if some objects are not reconstructed in the input image the network will not be able to represent them in the denoised image.

4.3 Learned Update Algorithm to Overcome Denoising Limits

In order to overcome the previously described denoising approach limits we implemented the Learned Update algorithm, the idea behind this algorithm is to use neural networks to iteratively update the reconstruction and reintroduce the information carried by the original data at each iteration as described in Chapter 2 section 2.3.2. Thanks to the reintroduction of the original data information if some details have not been well reconstructed in a given iteration they may be refined in the next one thus improving the final reconstruction.

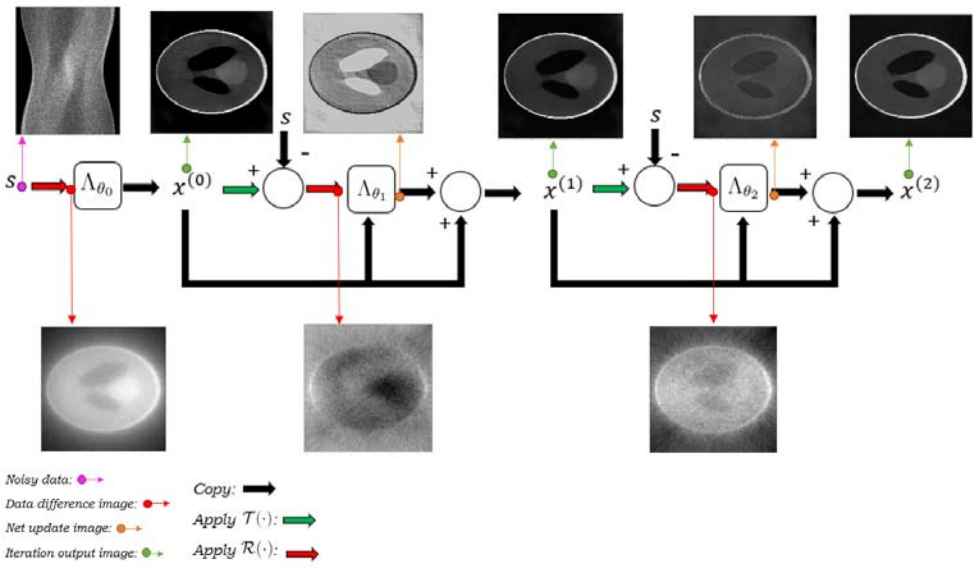


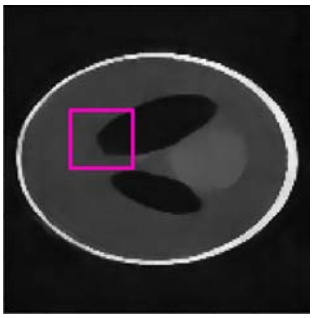
Figure 4.7: Three Iterations Learned Update Algorithm Workflow

Figure 4.7 shows the workflow of the three iterations LU algorithm. As we can see the reconstruction obtained after the first iteration is similar to the one obtained with the denoising approach since the Λ_{θ_0} architecture is exactly the same U-Net used in denoising problems and the only difference is in the Λ_{θ_0} input that in this algorithm is obtained with the adjoint of the forward operator instead of the MLEM_1 reconstruction. Then the iterative approach starts, the information carried by the original data is reintroduced and the first data difference image is obtained and used together with the first reconstruction to obtain the first update with the Λ_{θ_1} U-Net, this update is then added to the first iteration reconstruction to obtain the second one. As we can see in Figure 4.7 the first update is quite large and once added to the first iteration reconstruc-

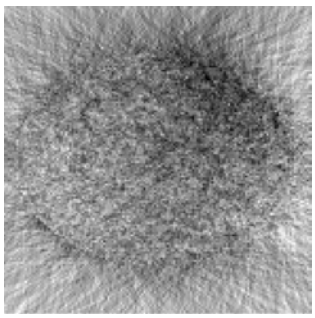
tion leads to a big improvement, this is due to the big values that are present in the data difference image since there is a big difference between the original data and those obtained from the first reconstruction. Then the process is repeated for another iteration, this time the update is not as strong as before and is more a refinement of the reconstruction of high contrast objects, again this is strongly related to the data difference image that this time has lower values and is more focused on high contrast objects since the difference between the second iteration reconstruction and the original data is not as big as it was in the previous iteration.

The reintroduction of the original noisy data information has a strong effect on each iteration reconstruction and leads to an improvement of the result until the third iteration. The reconstruction of small low contrast objects is still dependent on the particular realization of noise that is applied to the data, as it was for the denoising approach, but with the LU algorithm these details are represented more frequently than with the U-Net denoiser.

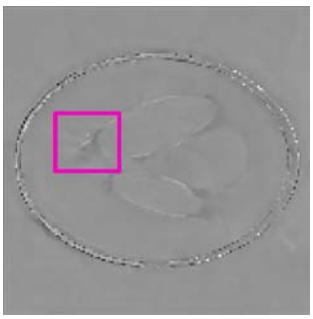
4.4 Learned Update Algorithm Stop Criterion



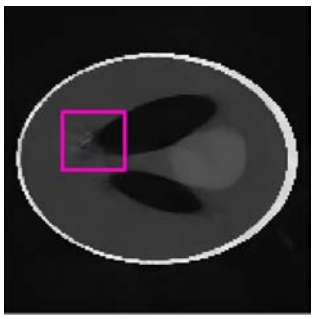
(a) LU 3 Iterations output



(b) 4^{th} iteration data difference



(c) 4^{th} iteration update



(d) LU 4 Iterations output

Figure 4.8: LU stop criterion

The best results with the LU algorithm are obtained by stopping after three iterations as shown in Chapter 3 section 3.2.1. The fourth iteration data difference image, shown in figure 4.8b, carries no information and is very noisy since the difference between the original noisy data and the data obtained from the three iteration LU reconstruction is almost only noise. The fourth image update, shown in Figure 4.8c, has an artifact near the terminal part of the top high contrast ellipse with a structure similar to the shapes that can be seen in the fourth iteration data difference image. This artifact is then introduced in the reconstruction when the update is added to the three iterations reconstruction as Figure 4.8d shows. It is important to note how this artifact was not present at all in the three iterations reconstruction shown in Figure 4.8a. The stop criterion for the LU algorithm is thus to interrupt the algorithm when there is no more relevant information coming from the data difference image, that is, at the third iteration in this context, since if we continue with more iterations noise related artifacts will start to appear thus reducing the reconstruction quality.

A five iteration LU algorithm has been tested as well and the corresponding reconstruction preserves the artifacts that are added in the fourth iteration but also tends to add more noise related artifacts that cover the whole image thus completely ruining the final result.

4.5 The Effect of Memory on the Learned Update Algorithm

The only difference between the Learned update and the Lerned Update with Memory algorithms is that in the LUM we allow each iteration U-Net to consider all the previous iterations reconstructions when computing the update to be applied to the current one.

Despite a small difference in the algorithm structure the difference in the behaviors of the two approaches is quite big. When the memory is applied the LU algorithm tends to lose the update workflow shown in Chapter 4 section 4.3 and starts to work more in parallel, each iteration will focus on different aspects of the image and all the different information coming from the various iterations will be mixed together from the last one. For example the first iteration will return an image that is related to the background noise, the second one will return an image carrying information related to high contrast objects and the third one will focus on low contrast objects, finally all these different images will be combined by the fourth iteration U-Net in order to obtain a

proper reconstruction.

Such a workflow does not work well with the LU algorithm and results are worse than those obtained with the memoryless algorithm as shown in Chapter 3 section 3.2.2. This is due to how the original noisy data are reintroduced in the process at each iteration, the LU algorithm idea works well when the result obtained from each iteration is a proper reconstruction because when this happens the data difference image of the next iteration will carry an information that is related to where the current iteration reconstruction is different from the ground truth image and this information may be useful for the following U-Net to compute an update that improves the reconstruction. But with the LUM algorithm the only proper reconstruction is obtained at the last iteration, all other iterations output are not reconstructions but images that carry information needed by the last U-Net to perform the reconstruction, the data difference images are thus not relevant since they are obtained from the difference between the original noisy data and the data obtained from an image that only shows some aspects of the whole reconstruction.

4.6 Learned Update on Both Image and Data Space: The Learned Primal-Dual Algorithm

The idea behind the Learned Primal-Dual algorithm is to use the Learned Update approach on both the data and the image space and exchange the information between the two spaces at each iteration. In this way the original noisy data are reintroduced at each iteration in a different way with respect to the LU algorithm since now they are given as input to a U-Net that will use them together with other inputs and will then give an output in the data space that will be projected in the image space and given as input to another U-Net. By doing this, we do not select a specific way to reintroduce the original noisy data information in the process, like computing the data difference and then obtain a difference image as done in the LU algorithm, but we allow our architecture to learn how to do so thus adding a degree of freedom.

With this new approach, it makes sense to use the memory since there is no more a data difference to be computed but the data space U-Nets will learn the best way to use all the information that will be given as input to compute the most useful output to be then used in the next image space U-Net. A memoryless version of the LPD algorithm has been tested as well but results were really poor. This is due to the fact that the LPD architecture is

very big even with a small number of iterations since for each iteration there are two U-Nets (one that works in the data space and one that works in the image space) and memory tends to have a regularization effect that makes the learning process more slow but also more stable thus improving the final result.

4.7 Learned Primal-Dual Algorithm Stop Criterion

The Learned Primal-Dual algorithm does not have a workflow similar to the Learned Update one and it is thus not possible to choose when to stop using the same criterion. The LPD algorithm workflow is more similar to the LUM one, with different iterations focusing on different aspects of the image and then all this information are combined together at the last iteration that performs the final reconstruction. A possible stop criterion could be to interrupt the algorithm when the outputs of different iterations starts to be similar one another since this would mean that the various iterations are already covering all the different aspects of the image. This is not the stop criterion that has been used in this context since the increasing computational time needed to train the whole architecture forced us to stop before the previously described behaviour started to manifest. For the LPD algorithm we stopped at three iterations since the training of a four iterations algorithm started to lead to acceptable results only after two weeks of training but the three iterations LPD results were still better and training time was lower as well.

4.8 On the Training of Neural Network-Based Iterative Algorithms

The most effective approach to train the neural network-based iterative algorithms that have been implemented in this project have been described in Chapter 2 sections 2.4.2-2.4.4. By training the architectures following those strategies we were able to allow the networks to work as intended while reducing the training time and being able to train multiple algorithms at once on a single Nvidia 1080Ti GPU. We also tried a different and more conventional training approach in order to check whether the Progressive Learning strategy is the only effective training strategy for our iterative algorithms. Following this approach we initialize with the *Xavier* initialization all the parameters of an iterative algorithm with a given number of iterations $I > 2$ and then start

the training procedure with the same number of training data and epochs used for the most effective approach.

Figure 4.9 shows the reconstruction obtained with the three iterations LU algorithm trained with this simpler approach, as we can see the result is very bad especially if compared to the three iterations LU trained with the PL strategy.

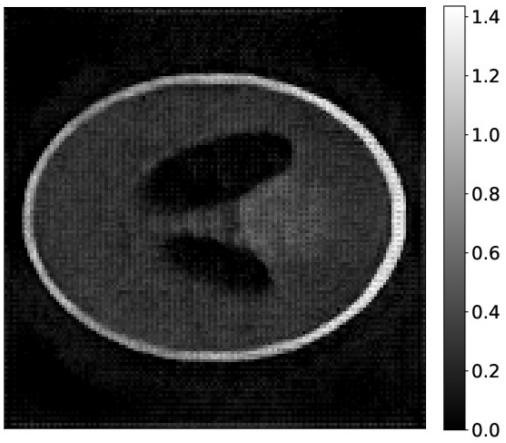


Figure 4.9: LU three iterations *Xavier* initialization result

The main issue that leads to this bad result is in the iterations updates, the U-Nets outputs, that are not relevant as Figure 4.10 shows.

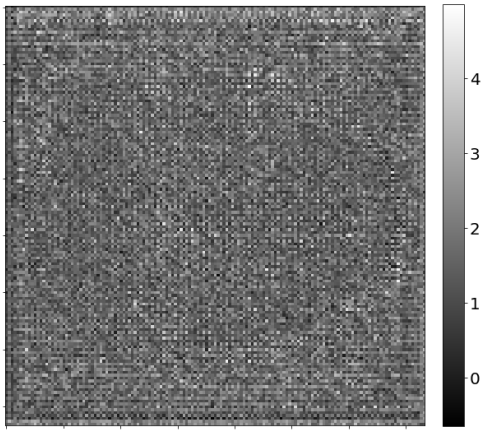


Figure 4.10: LU iteration update *Xavier* initialization

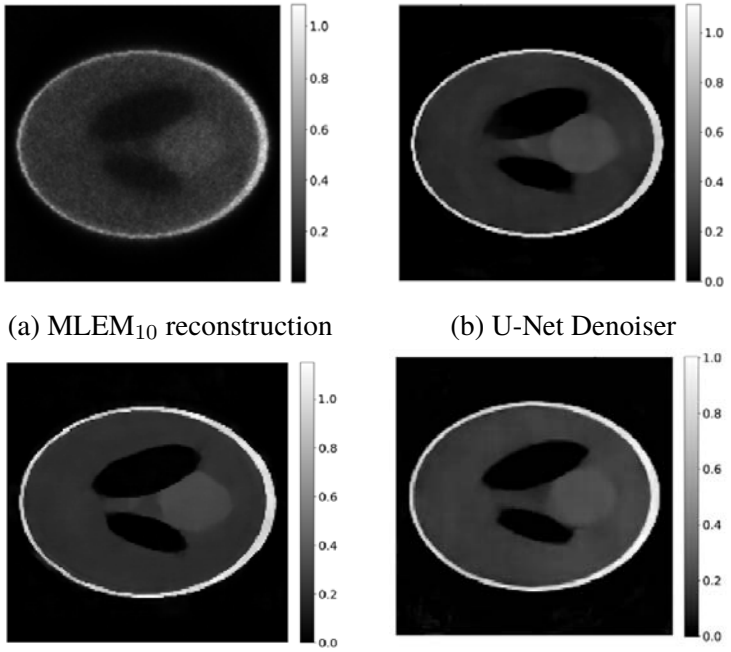
This is mainly due to the specific training approach that has been used, with the whole *Xavier* initialization approach the networks are not able to learn to

perform good reconstructions at each iteration and compute relevant updates but tend to discard information coming from all the iterations except the last one where the last U-Net finally attempts to perform a proper reconstruction. With the Progressive Learning strategy instead, the learning process is more controlled and bounded, the networks are thus able to properly learn how to perform well in the various iterations up to the last one. This may be due to the single iteration architecture complexity that is big enough to perform a reconstruction by itself, it would be interesting to use more simpler single iteration architectures and compare again the different training approaches.

A similar behaviour can be observed also when trying to apply this training approach to the Learned Primal-Dual algorithm.

The PL strategy proves to be a solid approach for the training of neural network-based iterative algorithms and outperforms simpler training approaches when used with the same amount of training data and epochs.

4.9 Denoising and Reconstruction Algorithms Performance on Synthetic Data



(a) MLEM_{10} reconstruction (b) U-Net Denoiser

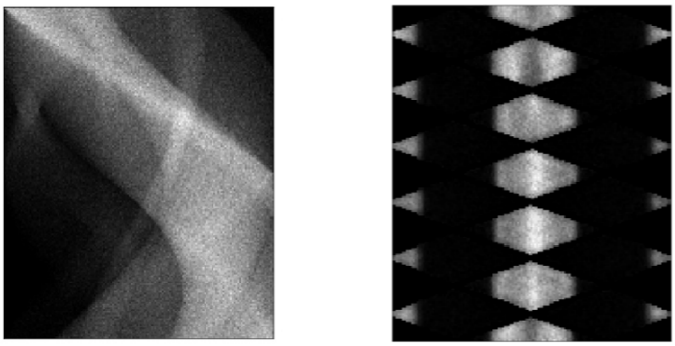
(c) LU reconstruction (d) LPD reconstruction

Figure 4.11: Denoising and Reconstruction synthetic data results

Both denoising and reconstruction approaches lead to a better looking result than the one obtained with the $MLEM_{10}$ reconstruction as Figure 4.11 shows. The U-Net denoiser is the least performing approach, as Figure 4.11b shows, since there are several artifacts near the terminal part of the dark ellipses in the denoised image. The LU reconstruction algorithm, Figure 4.11c leads to a very good result since there are no artifacts near the terminal part of the dark ellipses and the objects are very uniform. The best result is obtained with the LPD reconstruction algorithm, Figure 4.11d since there are no artifacts in the reconstructed image and the dynamic range is equal to the ground truth one that goes from 0 to 1, this is the only algorithm, among the implemented ones, that is able to reconstruct an image with the exact dynamic range and this is a key feature in PET image reconstruction.

4.10 From Synthetic to miniPET data

If we use the U-Net denoiser, the Learned Update algorithm and the Learned Primal-Dual algorithm that have been trained only with synthetic data on miniPET data results are surprisingly good as Figure 4.13 shows, especially considering the big difference between sinograms used for training and those obtained from a miniPET measure as Figure 4.12 shows.

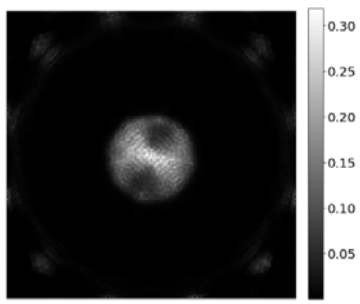


(a) Training set sinogram (b) miniPET sinogram

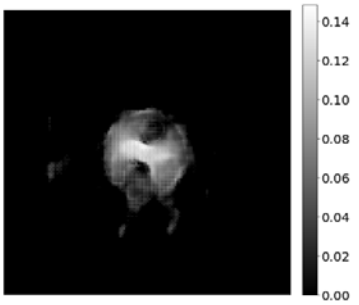
Figure 4.12: Training set and miniPET sinogram comparison

The U-Net denoiser is the least performing approach also in this context since the denoised image has some artifacts in the bottom part of the object as Figure 4.13b shows. Both LU and LPD reconstructions lead to a result that is comparable with the $MLEM_{10}$ reconstruction, Figure 4.13a. The LU reconstruction, Figure 4.13c is very smooth but removes the small borders

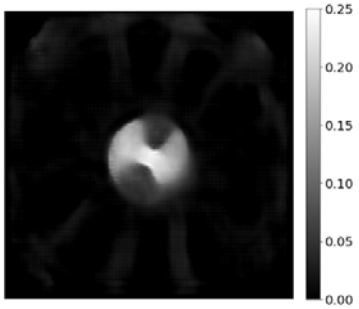
around the two cold spots of the IQ phantom. The best result is obtained with the LPD reconstruction, Figure 4.13d, since this algorithm is able to maintain the small features of the objects to be reconstructed while leading to a more smooth reconstruction with respct to the MLEM_{10} one.



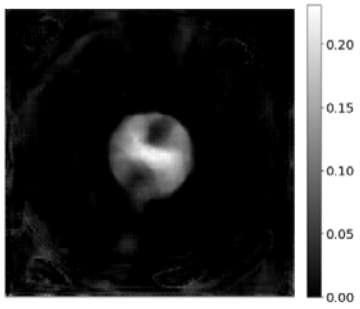
(a) MLEM_{10} reconstruction



(b) U-Net Denoiser



(c) LU reconstruction



(d) LPD reconstruction

Figure 4.13: Denoising and Reconstruction algorithms miniPET data results

The three iterations Learned Primal-Dual algorithm proves to be the best performing algorithm on both synthetic and miniPET data and is thus selected to be trained with miniPET data in order to improve its performance on the reconstruction of images starting from miniPET measurements.

4.11 On the Training of the Learned Primal-Dual Algorithm with miniPET Data

Figure 4.14 shows the results obtained with the three iterations LPD reconstruction algorithm trained with different approaches. The first image shows the result obtained by applying the LPD trained only with synthetic data on the miniPET test phantom data. The result is quite bad since a lot of shape artifacts appear near the various objects of the phantom. The image in the middle

shows the result obtained with the hybrid training of the LPD, this is the best one since objects are well reconstructed and there are no artifacts. The last image shows the result obtained with the miniPET only training of the LPD, also in this case the result is not really good since objects that are not really present appear in the top right portion of the mouse and there is a light shadow around the whole phantom.

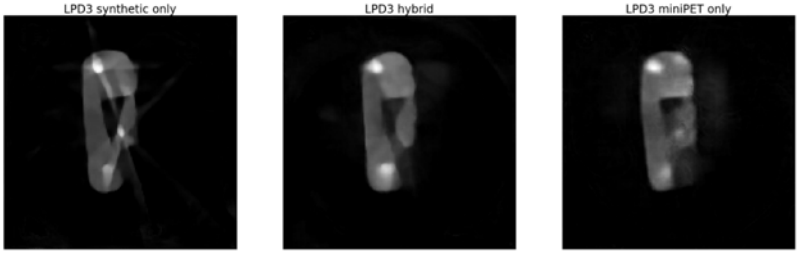


Figure 4.14: LPD 3 iterations: synthetic, hybrid and miniPET only training

The best training approach is thus the hybrid one, this is due to the low amount of miniPET data available and the high complexity of the LPD algorithm model. Even if we start from the parameters obtained with the synthetic training when continuing the training using only miniPET data the low amount of data available leads to overfit, this is why a light shadow appears around the object since training phantoms tend to have a more squared shape than the test one. The hybrid training instead has an effect similar to the regularization since the model will be forced to learn how to handle new miniPET data but at the same time it will need to remember how to handle also synthetic data thus being more constrained in the learning process and leading to a more general model that will give better results.

Chapter 5

Conclusions

Many are the conclusion that we can draw from this master thesis project. First we compared the performance of two different architectures for **PET Image Denoising**: an Encoder-Decoder and a U-Net. We concluded that the best performing architecture for our data was the U-Net but we also understood that the denoising approach has a big limitation: it is highly dependent to the initial reconstruction that will be given as input to the network. In order to overcome this limit, we implemented two **Neural Network-Based Iterative Algorithms** the Learned Update and the Learned Primal-Dual algorithms starting from the ideas of **Jonas Adler**[2]. The performance of these algorithms proved to be better than the simple denoising approach for **PET Image Reconstruction** with the Learned Primal-Dual being the most complex and promising one. We thus decided to extend the training of the **Learned Primal-Dual algorithm for the reconstruction of images from miniPET data**. We designed and 3D printed three training phantoms and one mouse like test phantom and then performed several measurements to build a training and test set. Finally, we developed two different approaches to be used to extend the training of the Learned Primal-Dual algorithm considering the low amount of data available: the miniPET only and hybrid training approaches. The hybrid training proved to be the best performing one and we were thus able to **improve the reconstruction** of images from miniPET data even **considering only a fraction of all measured data and a shorter acquisition window**.

Appendix A

Background

In this chapter an overview of the main topics involved in the master thesis project is given. We start with the explanation of how PET data are acquired and how images are reconstructed starting from these data. Then the deep learning framework is rigorously presented and finally we explain how images can be reconstructed from measured data using deep learning.

A.1 PET Imaging

A.1.1 PET Data Acquisition

In order to obtain a *Positron Emission Tomography* (PET) image a patient is injected with a positron-emitting radiopharmaceutical tracer. Various radioactive compounds can be employed as tracers, the most known and used one is fluorodeoxyglucose $[^{18}F]FDG$, a glucose analog with the positron-emitting radionuclide Fluorine-18 substituted for the normal hydroxyl group in the glucose molecule. Some commonly used radioactive tracers are reported in Table A.1.

Positrons are generated through the physical phenomenon of positron decay. When a nucleus undergoes positron decay one of its protons is converted into a neutron, at the same time a positron and an electron neutrino are emitted. An example of positron emission is shown in Figure A.1 with Fluorine-18 decaying into Oxigen-17 emitting a positron with a maximum energy of 0.634 MeV.

$$^{18}_{9}F \rightarrow ^{18}_{8}O + e^{+} + v_e \tag{A.1}$$

Table A.1: Some commonly used PET radionucleids, reprinted from [3] originally published in JNMT

Nuclide	Half-Life
^{18}F	110 min
^{11}C	20.3 min
^{13}N	9.97 min
^{15}O	125 sec

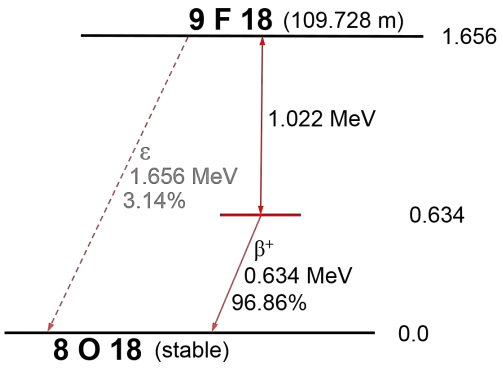


Figure A.1: Decaying scheme for $^{18}_{9}F$

PET imaging relies on the physical phenomenon of positron-electron annihilation.

$$e^+ + e^- \rightarrow \gamma + \gamma \tag{A.2}$$

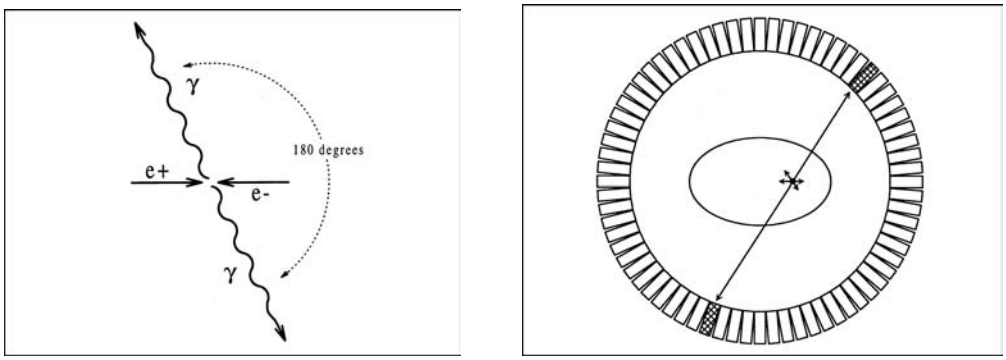
When a positron e^+ annihilates with an electron e^- two 512 keV gamma-ray photons γ moving in opposite directions are released as shown in Figure A.2a. A PET scanner consists of detectors arranged in a ring in order to be able to detect both gamma-ray photons emitted from the positron-electron interaction.

If two gamma-ray photons are simultaneously detected by two detectors of the PET scanner, we can infer that the annihilation has occurred along the line connecting the two detectors. Any line connecting two detectors is called *Line of Response* (LOR) [4].

The simultaneous detection of two photons is referred to as a *coincidence*, the number of coincidence events occurring along a given LOR will depend

on how much radioactivity there is along the line connecting the two detectors [3].

Figure A.2 summarizes what has been described in the previous paragraphs.



(a) (b)

Figure A.2: Positron annihilation (a) and coincidence detection (b), reprinted from [3] originally published in JNMT

Coincidence events can be divided in two main sets:

- True coincidences: the two detected gamma-ray photons originate from the same annihilation. True coincidences can be divided in two subsets:
  - Primary: after the annihilation the two gamma-rays travel along a line towards the detectors,
  - Scattered: after the annihilation photons scatter prior to detection and hence arrive at the wrong point.
- Random coincidence: the two detected gamma-ray photons originate from different annihilations and two independent detections occur by chance at the same time.

Only true coincidence events are to be used for image reconstruction. However also true coincidence are not perfect because of:

- Annihilation acollinearity: The paths of the two radiation photons are not exactly collinear, the slight angular deviation from 180° due to electron-positron momentum [5] causes a separation of the LOR from the true electron-positron emission point [6],

- Positron range: the positron travels for some millimeters from the emission point before annihilation, the emission point and the annihilation point are thus different. The distance that the positron travels depends on its energy and the density of the material the positron travels through (i.e. the tissue) [7].

A.1.2 PET Image Reconstruction Theory

The number of coincidence events detected in a given acquisition window is proportional to the integral of the tracer concentration along the LOR, we can thus write:

$$\sum_{LOR}(events) \propto \int_{LOR} f(\mathbf{x})d\mathbf{x} \tag{A.3}$$

Where:

- $\sum_{LOR}(events)$: Sum of the number of coincidence events occurred along a given LOR,
- $f(\mathbf{x})$: activity concentration at position $\mathbf{x} = (x, y)$.

Even assuming to have data corrected for the coincidences giving wrong spatial positions described in the previous section, equation A.3 is only true on average due to the stochastic nature of positron emission and photon detection. It is thus better to assume the sum of coincidence events on a LOR to be a realization of a random variable with a Poisson probability density function, so that the line-integral in A.3 represents the expected value that can be computed from the underlined probability density function [8]. We can thus rewrite Equation A.3 as:

$$E\left[\sum_{LOR}(events)\right] = E\left[c\int_{LOR} f(\mathbf{x})d\mathbf{x}\right] \tag{A.4}$$

Where:

- c: proportionality constant.

Equation A.4 considers only counts of coincidence events measured along a single LOR, in a PET scanner more than one LOR is present, in particular for a scanner with N detectors the maximum number of LORs that can be considered is $\frac{N^2}{2}$. The model described in Equation A.4 can thus be generalized to consider counts occurring along different LORs.

In order to identify a LOR in the space we need to parametrize it, two parameters are introduced to do so:

- ρ: shortest distance between LOR and center of PET scanner gantry,
- θ: angle of orientation of the LOR.

An example of LOR parametrization is reported in Figure A.3.

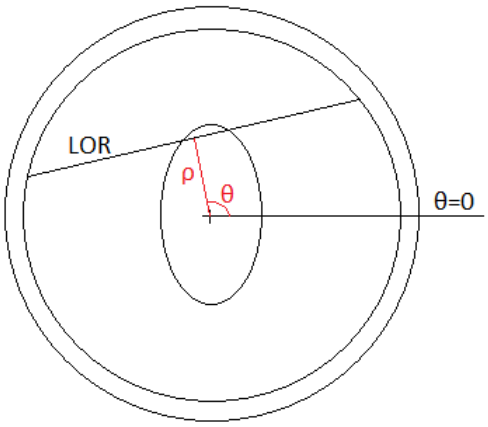


Figure A.3: Example of LOR parametrization

With this parametrization we can now write the measure obtained from any couple of PET detectors that can be connected by a line parametrized by ρ and θ as:

$$E\left[\sum_{LOR_{(\rho,\theta)}}(events)\right] = E\left[c\int_{LOR_{(\rho,\theta)}} f(\mathbf{x})d\mathbf{x}\right] \tag{A.5}$$

Where:

- $LOR_{(\rho,\theta)}$: LOR parametrized by ρ and θ,
- $\sum_{LOR_{(\rho,\theta)}}(events)$: Sum of the number of coincidence events occurred along a given LOR.

Equation A.5 is a simple definition of the *Forward Model* for PET imaging, such model can be used to obtain the data measured by any couple of PET scanner detectors if we know the activity distribution $f(\mathbf{x})$. In the real world more parameters govern the system physics and the forward model becomes

thus more complex. However in reality, we face the inverse problem since we are able to measure the counts but we do not know the activity distribution. The purpose of the inverse problem is to use the measured data along multiple LORs to reconstruct the image.

From a theoretical point of view, the forward model is invertible and given measures along an infinite number of lines we can reconstruct the image. This is not possible to be implemented in practice since inverse problems like this, in which the unknown $f(x)$ is a function, are frequently ill-posed, similar boundary conditions lead to very different results, and this problem certainly fits into the ill-posed category since the measures are affected by noise. Different realizations of noise applied to the same data will thus lead to very different reconstructed images.

A.1.3 PET Image Reconstruction Algorithms

In order to overcome the impossibility of a perfect inversion of the forward model, many algorithms have been developed to reconstruct images from measured data.

Often data are rearranged into sinograms before being given as input to these algorithms. A sinogram is a graph where the angle of orientation of a LOR θ is plotted on the y-axis and the shortest distance between the LOR and the center of the gantry ρ is plotted on the x-axis, the number of coincidence events occurring along a given LOR is associated to the couple (ρ, θ) that parametrize the LOR connecting the detectors and plotted in the graph. A fixed point in a rotating framework describes a sinusoidal wave, as shown in Figure A.4, hence the name sinogram. A more complicated object will cover many pixels, and thus, its sinogram will consist of a large number of sinusoidal waves. A sinogram of a PET slice through the brain and its corresponding transverse reconstruction are shown in Figure A.5

If we fix a value for θ and let ρ vary we obtain a *Projection* because values of the sinogram selected in this process represent detections acquired along parallel LORs at the angle that corresponds to the fixed θ, each selected pixel value is the sum of all of the events acquired along the corresponding LOR [4].

The most simple algorithm for image reconstruction is the *Back-Projection* algorithm. Following the back-projection approach, the measured signal is projected back across the direction of the LOR that generated the considered projection, once all the projections have been back-projected all the obtained images are summed and divided by the number of projections available. As

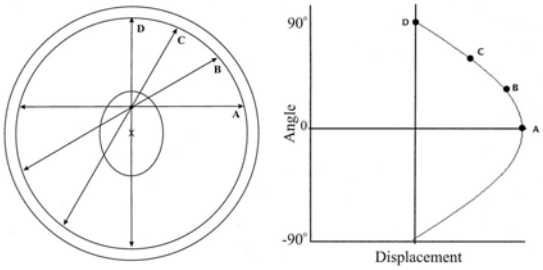


Figure A.4: LORs and corresponding sinogram, reprinted from [4] originally published in JNMT

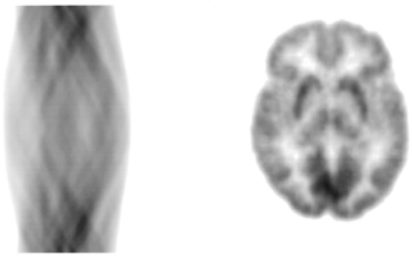


Figure A.5: Sinogram of a brain PET image, reprinted from [4] originally published in JNMT

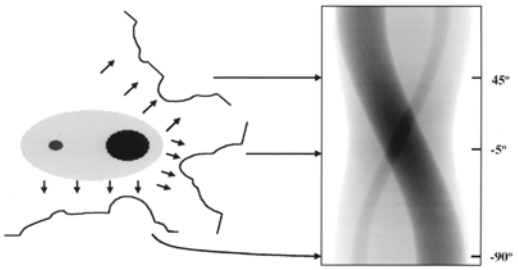


Figure A.6: Projections and corresponding rows in the sinogram, reprinted from [4] originally published in JNMT

Figure A.7 shows the result improves if the number of back-projections is increased, however even using many back-projections the result is not good, the output image is quite blurry. A single point in the true image is reconstructed as a circular region that decreases in intensity away from the center [9].

In order to reduce the blurring encountered in the back-projection approach, the *Filtered Back Projection* algorithm can be used. Following this approach each projection is filtered. The filter definition and the mathematical background behind such a filter choice can be found in [10]. before being back-

projected, after this filtering step the approach is the same as the previous one as shown in Figure A.8.

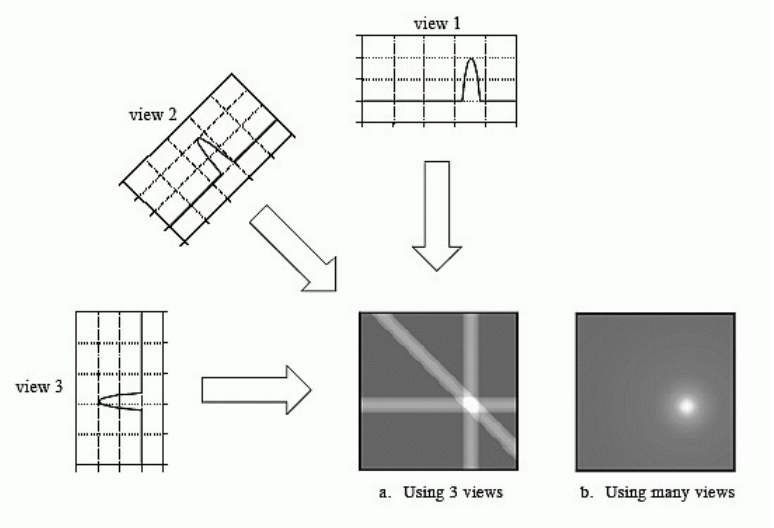


Figure A.7: Back-projection reconstruction, reprinted from [9]

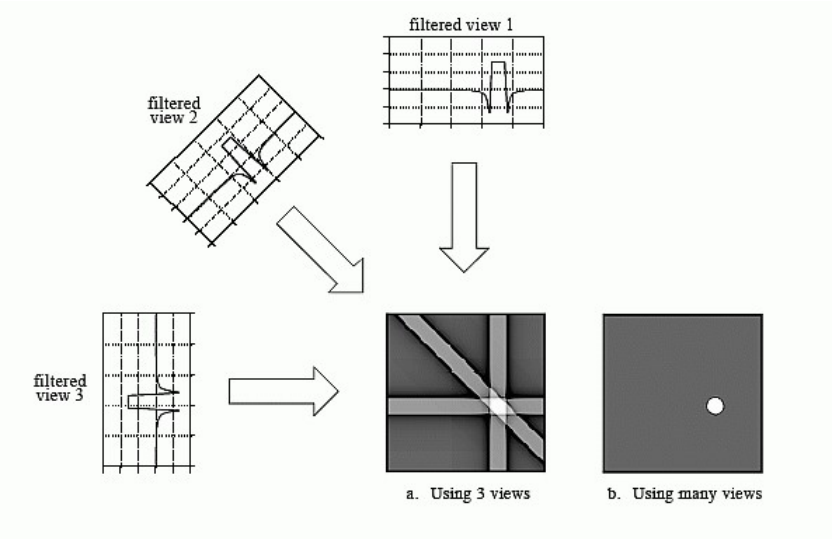


Figure A.8: Filtered back-projection reconstruction, reprinted from [9]

The simple back-projection algorithm is never used since it leads to poor results, the filtered back-projection, is the inverse of the forward model and gives good results even with a finite number of projections in absence of noise but the quality of the reconstruction deteriorates rapidly when the noise level

increases. For this reason the filtered back-projection algorithm is used as the standard algorithm for *Computerized Tomography* (CT) image reconstruction but gives less satisfactory results for PET image reconstruction where the data noise level is much higher than in CT data.

Iterative methods lead to better reconstruction results for PET imaging when using enough iterations. The diagram in Figure A.9 summarizes the steps of an iterative algorithm. The procedure starts with an initial guess, often all the image pixels are set equal to zero. Projections are computed from the image that is being reconstructed and their values are compared with the measured projections along the same directions. If there is a difference between the estimated and measured projections, corrections are made to the estimated image in order to improve it and a new iteration is performed to assess the convergence between estimated and measured projections. Iterations are continued until an agreement between the two sets of projection is achieved [11]. Many algorithms have been designed following this framework and the main difference between them lies in the update rule.

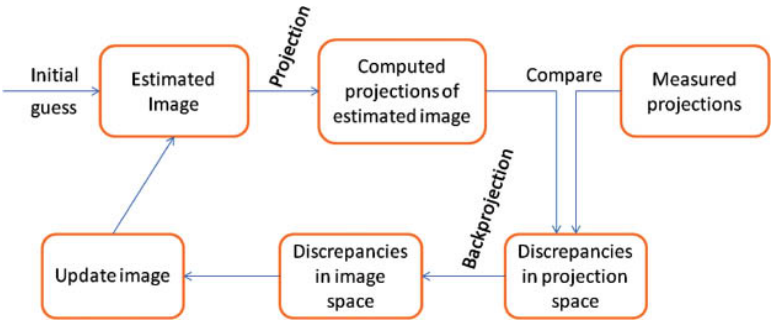


Figure A.9: Flow chart of the iterative image reconstruction method, reprinted from [11] use permitted under the Creative Commons Attribution License CC BY

The *Maximum Likelihood Expectation Maximization*, or shortly MLEM, algorithm follows the iterative methods idea and its update rule can be described as follows [12]:

$$\lambda_j^{k+1} = \frac{\lambda_j^k}{\sum_i^m C_{i,j}} \sum_i^m \frac{C_{i,j}}{\sum_j^m C_{i,j} \lambda_j^k} \tag{A.6}$$

Where:

- λ_j^k: value of reconstructed image at pixel j for the k-th iteration,

- $C_{i,j}$: probability of detecting an emission from the pixel j in the i^{th} value of the projection,
- m: total number of projection values.

At each iteration k all the image pixels are updated according to A.6, $C_{i,j}$ values are computed from the sinogram acquired by the PET scanner. This algorithm converges to the maximum likelihood estimate of a probability distribution function from the observed data [13]. In this algorithm, the measured emission data is assumed to be a spatially dependent Poisson model [12]. A nice feature of this algorithm is that it has a simple multiplicative update step and is thus fast and easy to implement.

A.1.4 PET Image Quality Evaluation

The quality of a reconstructed image can be assessed with two different approaches:

- Qualitative evaluation: An observer looks at the image and gives a personal evaluation of the quality of the reconstruction, aspects like resolution, dynamic range, and absence of artifacts may be considered. This kind of approach is highly subjective,
- Quantitative evaluation: Figures of merit are computed from the image pixel values. This kind of approach is objective but multiple figures of merit may be considered and statistical validation of the results should be carried on in order to obtain a reliable result.

Figures of merit that are often used for image quality evaluation are the *Peak Signal-to-Noise Ratio* (PSNR) and the *Structural Similarity Index Method* (SSIM).

The PSNR is used to evaluate the amount of noise that corrupts an image or, in general, a signal. The PSNR is defined as follows:

$$\text{PSNR} = 10 \log_{10} \left(\frac{\max^2(I)}{\text{MSE}} \right) \tag{A.7}$$

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{A.8}$$

Where:

- m: number of rows of the image,

- n: number of columns of the image,
- I: noise-free version of the image,
- K: noisy image,
- $\max(I)$: maximum value of the noise-free version of the image.

Images with high PSNR are less noisy and thus better reconstructed than images with low PSNR as Figure A.10 shows.

The SSIM is used to measure the similarity between two images X and Y, it is a perception-based model that considers image degradation as perceived change in structural information, while also incorporating important perceptual phenomena such as luminance and contrast, it can be defined as follows [14]:

$$\text{SSIM} = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{A.9}$$

Where:

- μ_x: average value of image X,
- μ_y: average value of image Y,
- σ_x^2: variance of image X,
- σ_y^2: variance of image Y,
- σ_{xy}: covariance of images X and Y,
- C_1 and C_2: constants to avoid computational issues when $(\mu_x^2+\mu_y^2+C1)$ or $(\sigma_x^2 + \sigma_y^2 + C_2)$ is close to zero.

The SSIM is a number that goes from -1 to 1, SSIM $= 0$ means no similarity between the images and SSIM $= 1$ means that the two images are identical. Images with SSIM closer to 1 are better as Figure A.10 shows.

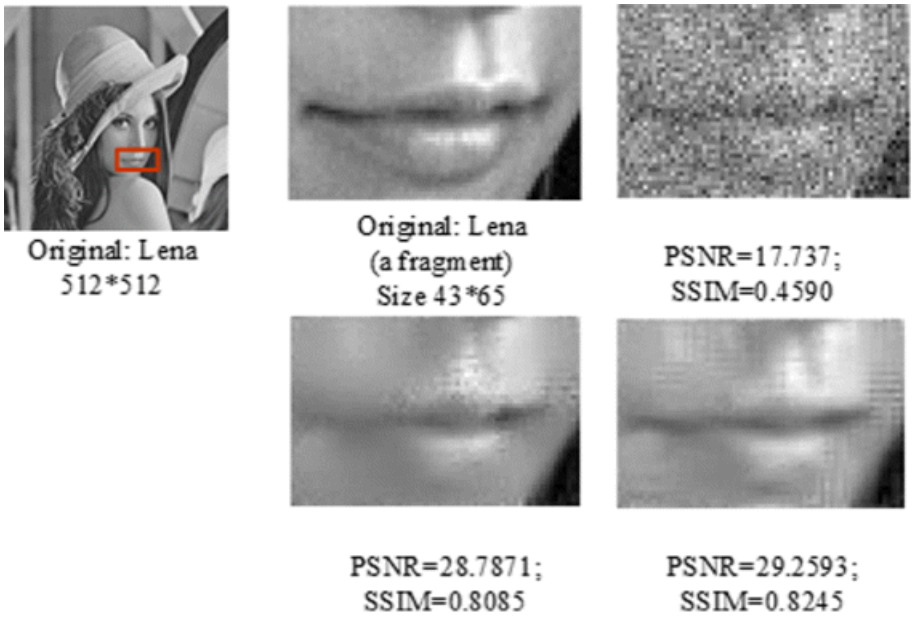


Figure A.10: PSNR and SSIM, different levels of noise, reprinted from [15], use permitted under the Creative Commons Attribution License CC BY

A.2 Machine Learning Fundamentals

A.2.1 Machine Learning Supervised Framework

Supervised machine learning methods rely on the following formal learning model.

The main components of a formal model aimed to describe supervised learning tasks are [16]:

- Model class: A set of mathematical models H,
- Domain set: An arbitrary set of objects X that we give as input to a model $h \in H$,
- Target set: Another arbitrary set of objects Y that are the targets that we want to reach with the model $h \in H$ when given objects from X as input,
- Training data: $S = ((x_1, y_1), \cdots (x_m, y_m))$ is a finite sequence of pairs in $X \times Y$, a sequence of domain set objects and corresponding target set objects,
- Measure of success: A function, often called "Loss function" L that measures how good the model $h \in H$ is at finding the $y_i \in Y$ when given the corresponding $x_i \in X$ as input,
- Learning algorithm: An algorithm A that when given the training data as input outputs the best performing model $h \in H$ according to the measure of success L.

As an example, the simple linear regression problem can be cast as a learning problem where: the model class is the equation of a line $y_i^{'} = ax_i + b$ described by its parameters a, b, the domain set X is a subset of $\mathbb{R}$, the target set Y is a subset of $\mathbb{R}$ as well, the training data are m known couples of points $(x_i, y_i) \quad i = 1, \cdots, m$, the measure of success is the second power of the difference between the true value y_i and the predicted value, the model output y_i', and finally the algorithm is a minimization algorithm that finds the parameters $\hat{a}, \hat{b}$ of the line that leads to the minimum loss.

$$\hat{a}, \hat{b} = \underset{a,b}{\operatorname{argmin}} \sum_{i=1}^{m} [y_i - ax_i - b]^2 \tag{A.10}$$

A.2.2 The Neural Network Model Class

An artificial neural network is a model of computation inspired by the structure of neural networks in the brain, it consists of a large number of basic computing devices called *Neurons* that are connected to each other in a complex communication network [16].

A neural network can be described as a directed graph G whose nodes V correspond to neurons and edges E correspond to links between them. Each neuron receives as input a weighted sum of the outputs of the neurons connected to its incoming edges.

$$G = (V, E) \tag{A.11}$$

Weights are assigned to each edge according to a weight function ω:

$$\omega : E \to \mathbb{R} \tag{A.12}$$

Every single neuron is modeled as a simple scalar function σ often referred to as the *Activation Function* of the neuron:

$$\sigma : \mathbb{R} \to \mathbb{R} \tag{A.13}$$

The network is organized in *Layers*, each layer contains a subset of nodes such that every edge in E connects some node in the layer $j-1$ to some node in the j^{th} layer. The first layer of the network is called *Input Layer*, the last layer is called *Output Layer* and all the other layers are called *Hidden Layers*. A neural network is *Fully Connected* if all nodes of layer $j-1$ are linked to all nodes of layer j.

A summarizing example of a neural network graph is shown in Figure A.11 and its mathematical representation is reported in Equation A.14 where $\alpha_i, \beta_i, \gamma_i \in \mathbb{R}$ are the graph weights:

$$y = \left(\sum_{i=1}^{4} \alpha_i \sigma(\beta_i x + \gamma_i) \right) + C \tag{A.14}$$

According to the *Universal Approximation Theorem* proved in [17], a very simple neural network with a single hidden layer containing a finite number of neurons like the one just described can represent a wide variety of functions.
Theorem 2.1, Universal approximation theorem: Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded, and continuous function (the previously described activation function). Let I_d be the m-dimensional unit hypercube $[0, 1]^d$. The space of real-valued continuous functions on I_d is denoted by $C(I_d)$. Then,

given any $\epsilon > 0$ and any function $f \in C(I_d)$, there exist an integer N, real constants $\alpha_i, \gamma_i \in \mathbb{R}$ and real vectors $\beta_i \in \mathbb{R}^d$ for $i = 1, \ldots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} \alpha_i \sigma(\beta_i^T x + \gamma_i) \tag{A.15}$$

as an approximate realization of the function f; that is,

$$|F(x) - f(x)| < \epsilon \tag{A.16}$$

for all $x \in I_m$.

Considering A.15 as the function of a simple neural network and f as the true function that links the domain set to the target set, Theorem 2.1 states that the learned $F(x)$ well approximates the true function $f(x)$, this shows why neural networks are so powerful for learning tasks and can be used in a wide set of applications.

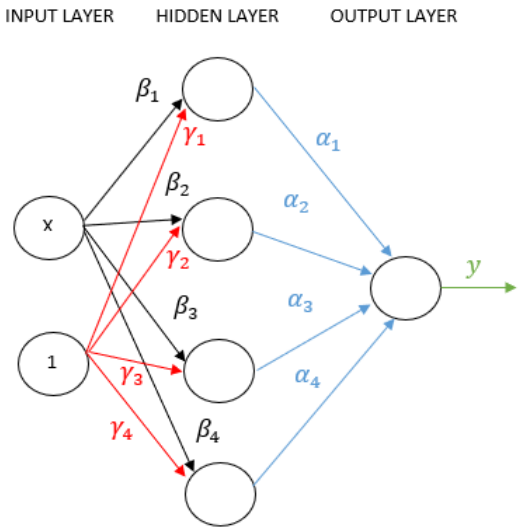


Figure A.11: Graph of a Fully Connected Neural Network, adapted from [18]

A.2.3 Neural Network architectures

When dealing with learning problems that involve images the most used neural networks are CNNs, short for *Convolutional Neural Networks*. As shown in Figure A.12 convolutional neural networks are not fully connected networks, neurons of the input layer are the pixels of the image and neurons of the hidden layers can be seen as pixels of filtered versions of the previous layer image since the weights of the edges have the role of convolution kernels weights. According to the specific learning task also some fully connected layers can be present inside the convolutional network architecture after some convolutional layers.

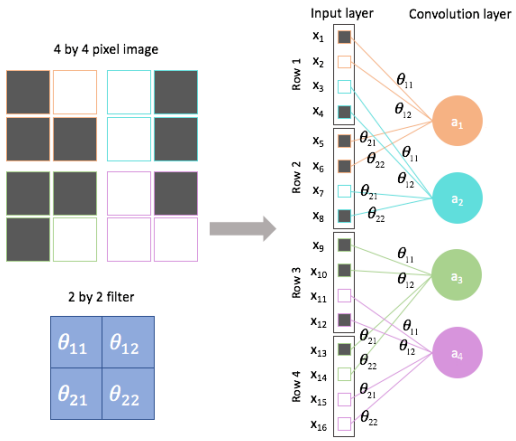


Figure A.12: Graph of a Simple Convolutional Layer

As Figure A.13 shows, convolutional neural networks can thus be seen as a series of filters applied one after the other starting from the input image, the parameters of such filters are learned during the network training when the output of the network is compared to the target associated to the input.

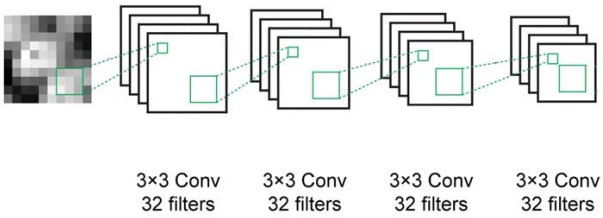


Figure A.13: CNN as a Series of Filters, reprinted from [19] use permitted under the Creative Commons Attribution License CC BY

A particular CNN architecture is the U-Net. A U-Net is characterized by two paths, the contracting path and the expansive path [20]. The contracting path follows the typical architecture of a convolutional neural network. It consists of the repeated application of two 2D 3x3 convolutions, each followed by a batch normalization and a rectified linear unit or ReLU activation function which graph is showed in Figure A.14.

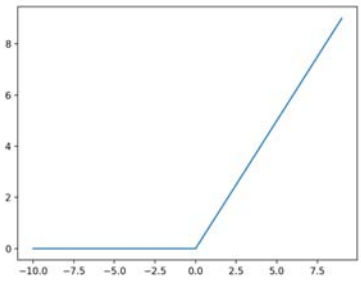


Figure A.14: ReLU activation function

After the two 2D convolutions, a 2D max-pooling operation is performed and the number of feature maps is doubled. The pooling is done with a 2x2 sliding window with stride 2 to keep only important features in the filtered image while downsampling it to reduce the model complexity. Every step in the expansive path starts with a 2D upsampling of the feature map, needed to return to the input image size after the pooling operations, and is performed with a 2x2 2D up-convolution that also halves the number of feature channels. After the upsampling a concatenation with the correspondingly feature map from the contracting path, two 2D 3x3 convolutions, each followed by a batch normalization and a ReLU activation function are applied. At the final layer, a 2D 1x1 convolution is used to map each 32 component feature vector to the single-channel output image. Between the contractive and expansive paths, two 2D 3x3 convolutions each followed by a batch normalization and a ReLU are applied without pooling or upsampling operations. An example of a 2D U-Net architecture with three layers of depth is shown in Figure A.15. The U-Net was first proposed for image segmentation, but by changing the number of output channels to one, it can also be used for post-processing as was done in [21]. An Encoder-Decoder architecture can be obtained by removing the concatenations between contracting and expansive paths feature maps as shown in Figure A.16, by doing so the two paths become independent but it will be harder to reconstruct the image in the expansive path since, in each layer, there is no information coming from the contracting path.

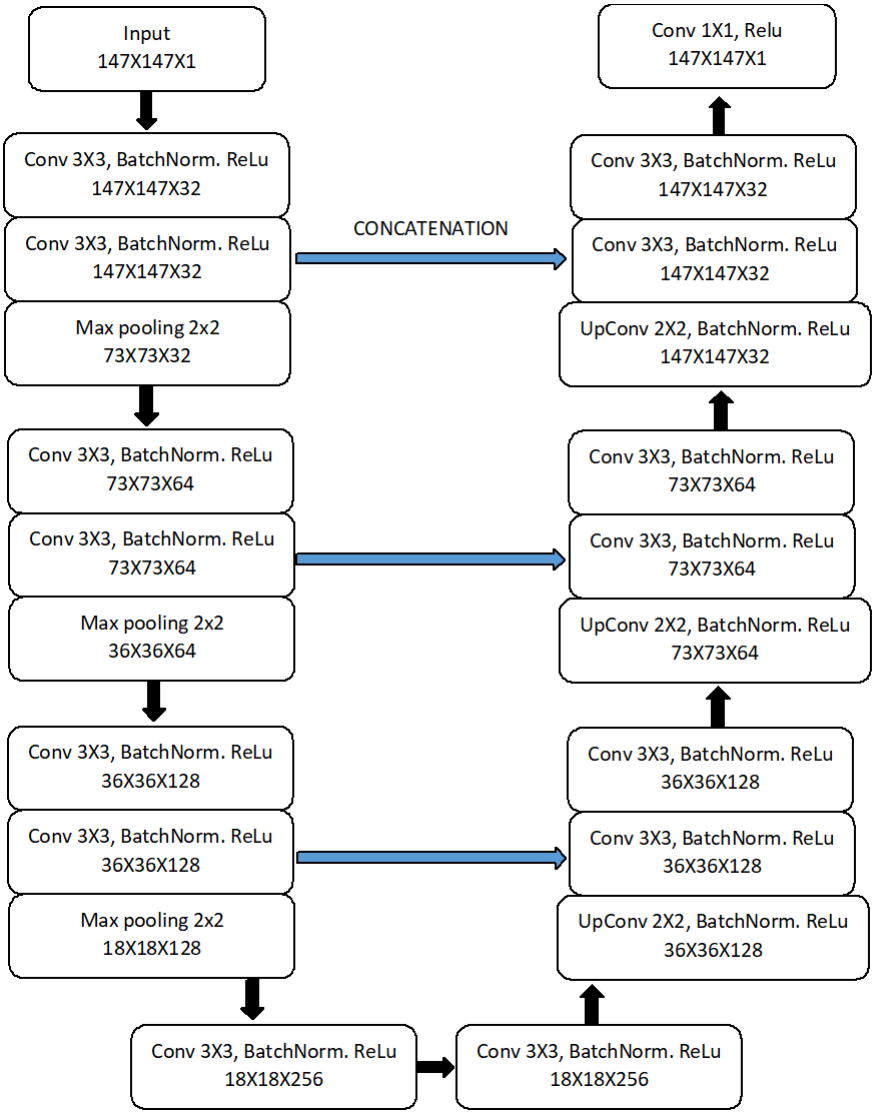


Figure A.15: U-Net architecture, 3 layers depth

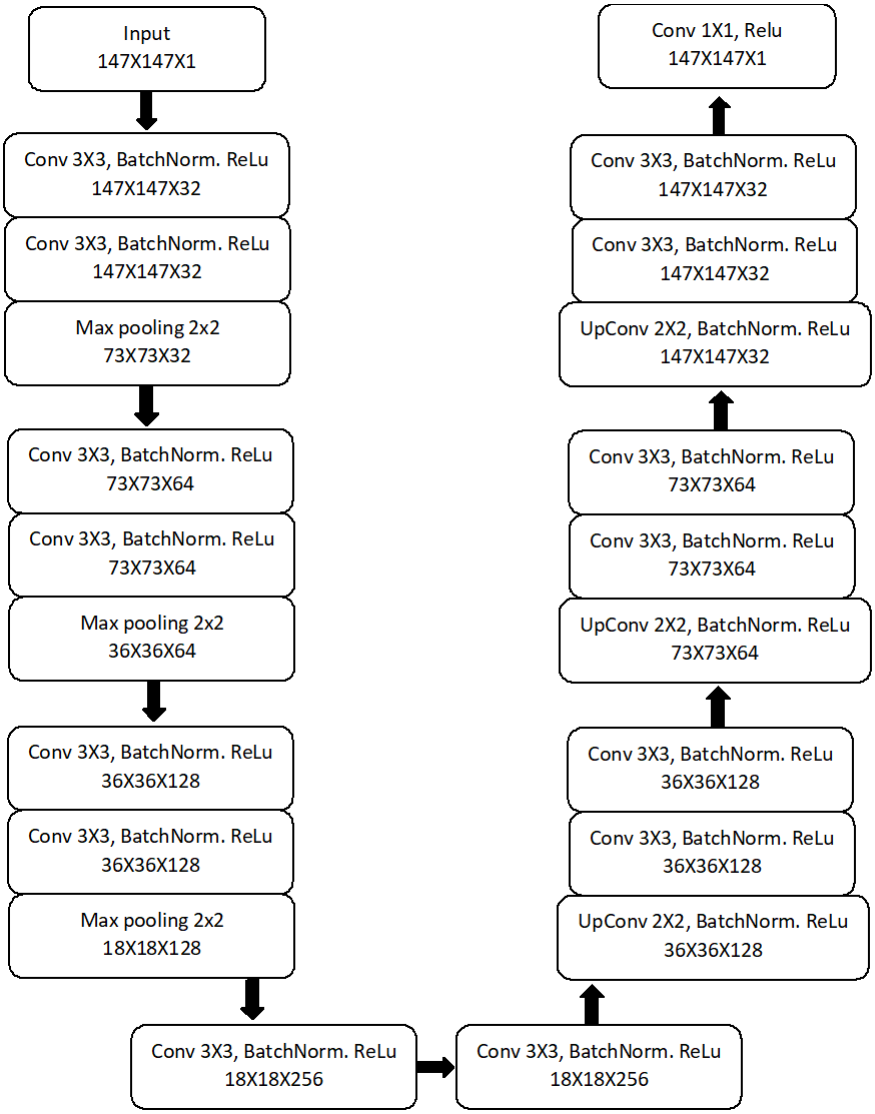


Figure A.16: Encoder-Decoder architecture, 3 layers depth

A.2.4 Loss Functions as a Measure of Success

Loss functions are used to evaluate the performance of a model in a learning task, consider any model class H and some domain Z (considering the learning framework introduced before we may have $Z = X \times Y$) let l be any function from $H \times Z$ to the set of nonnegative real numbers:

$$l : H \times Z \to \mathbb{R}_+ \tag{A.17}$$

We call such functions *Loss Functions* [16].

We now define the *Risk Function* to be the expected loss of a model $h \in H$, with respect to a probability distribution D over the domain Z:

$$L_D(h) = E_{z \sim Z}[l(h, z_i)] \tag{A.18}$$

That is, we consider the expectation of the loss of the model h over an infinite amount of objects z picked randomly according to D from Z [16]. Such a definition of risk is true only in theory since, in reality, we do not have an infinite amount of data available, we thus need to introduce the *Empirical Risk*. The empirical risk is the expected loss over a given sample $S = (z_1 \cdots z_m) \in Z^m$, like the previously introduced training data set:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} l(h, z_i) \tag{A.19}$$

The empirical risk is considered as a measure of success for the model h on the whole training data set S according to the loss function l.

Many functions can be used as a loss but there is not a loss that is good for all learning tasks, choosing a proper loss function is key to achieve a good model performance. Some of the most known studied and used loss functions are:

- Square loss or L^2 loss: Considering z as a pair $(x, y) \in X \times Y$ this loss function is defined as [22]:

$$\mathrm{L}^2(h, (x, y)) = \frac{1}{2}[h(x) - y]^2 \tag{A.20}$$

- Absolute loss or L^1 loss: Considering z as a pair $(x, y) \in X \times Y$ this loss function is defined as [22]:

$$\mathrm{L}^1(h, (x, y)) = |h(x) - y| \tag{A.21}$$

- Smooth L^1 loss: Considering z as a pair $(x, y) \in X \times Y$ this loss function is defined as [22]:

$$L^1_{smooth}(h, (x, y)) = \begin{cases} \frac{1}{2}[h(x) - y]^2 & \text{if} \quad |h(x) - y| < 1 \\ |h(x) - y| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{A.22}$$

Figure A.17 shows a plot of the three loss functions defined above.

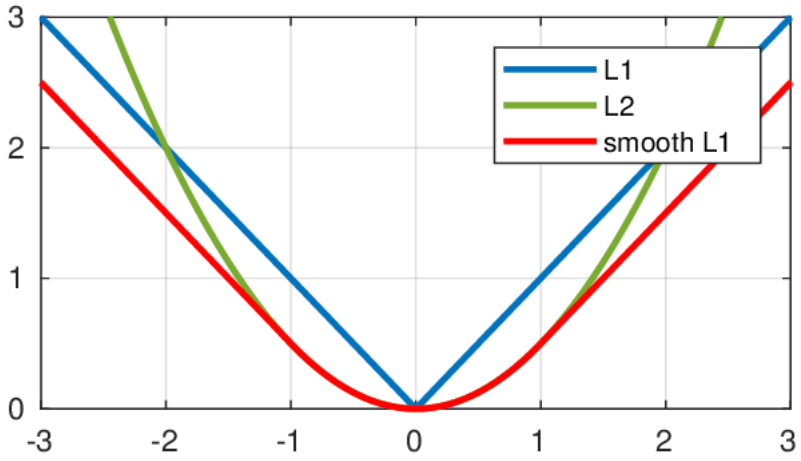


Figure A.17: L^2 loss, L^1 loss and L^1_{smooth} loss, reprinted from [22] use permitted under the Creative Commons Attribution License CC BY

L^2 loss function considers the squared differences between the estimated and true target values, L^1 loss function considers the absolute differences instead. L^2 error will be much larger in the case of outliers compared to L^1 because the difference between an incorrectly predicted target value and true target value will be quite large and squaring it will make it even larger. As a result, L^1 loss function is more robust and is generally less affected by outliers. On the contrary L^2 loss function will try to adjust the model according to these outlier values, even at the expense of other samples. Hence, L^2 loss function is highly sensitive to outliers in the dataset. On the other hand L^2 loss is more precise and better in minimizing prediction errors since it leads to fewer oscillations during updates when the error is small. L^1_{smooth} loss combines the advantages of L^1 and L^2 by keeping the best-performing fragments of the two loss functions.

A.2.5 Optimization Algorithms

The best performing model $\hat{h} \in H$ is chosen as the one that minimizes the empirical risk, that is:

$$\hat{h} = \underset{h \in H}{\operatorname{argmin}} L_s(h) \tag{A.23}$$

This procedure is called *Empirical Risk Minimization*.

Being the empirical risk a function that depends on the model h parameters vector $\mathbf{x}$, the previous problem can be cast as an optimization problem where given a function with some unknown parameters we estimate the parameters that lead to the minimum value of that function.

The most known approach for minimizing a generic differentiable function $f(\mathbf{x})$ is the *Gradient Descent*. The gradient of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ at $\mathbf{x} \in \mathbb{R}^d$, denoted as $\nabla f(\mathbf{x})$ is the vector of partial derivatives of f, namely, $\nabla f(\mathbf{x}) = (\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}[1]}, \cdots, \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}[d]})$. Gradient descent is an iterative algorithm. We start with an initial value of $\mathbf{x}$, for example, $\mathbf{x}^{(0)} = 0$, then, at each iteration, we take a step in the direction of the negative of the gradient at the current point. Considering $t \geq 0$ as the iteration number the update step is thus:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)}) \tag{A.24}$$

Where $\alpha > 0$ is a parameter that determines how much we move along the negative gradient direction. Since the gradient points in the direction of the greatest rate of increase of f around $\mathbf{x}^{(t)}$, the algorithm makes a small step in the opposite direction, thus decreasing the value of the function [16].

The gradient descent algorithm works well when:

- The function $f(\mathbf{x})$ is convex so there is only one global minimum point. If this doesn't happen the algorithm may lead us towards a local minimum point and becomes really sensitive to the choice of the initial guess $\mathbf{x}^{(0)}$,
- The parameters of the model are not a lot, that is small d, because if the number of parameters is high the number of derivatives to compute is high too and performing more computations requires more time and it must be done for each iteration,
- The training data are not a lot, that is small m, because when computing $L_s(h)$, that is $f(\mathbf{x})$ if we consider our learning framework, losses computed for all the data are considered and with more data doing this requires more time and this must be done for each iteration.

Considering the neural networks model class all these three points are not true: even the simple neural network model function described in A.14 is highly nonconvex, the parameters of the model are often a lot even for simple networks and the number of training data must be big in order to obtain good results.

An alternative algorithm that works well with neural networks is the *Stochastic Gradient Descent* or SGD. The only difference with respect to the standard gradient descent is in how we compute $L_S(h)$, in the stochastic gradient descent we use $L_i(h)$ instead where:

$$L_i(h) = l(h, z_i) \tag{A.25}$$

with z_i randomly extracted from the training data set S. At each iteration of the algorithm a sample is randomly extracted from the training data set, the gradient of the loss is computed and the parameters vector $\mathbf{x}$ is updated according to A.24. With the gradient descent approach, the direction is a random vector and only its expected value at each iteration is equal to the gradient direction [16]. This approach eases the computational complexity of the standard gradient descent and since there is some randomness involved it may happen that instead of always going towards the minimum point we go backward, as Figure A.18 shows, this may be good with non convex functions because by going backward we may go away from a local minimum and arrive near the global one on the next iteration.

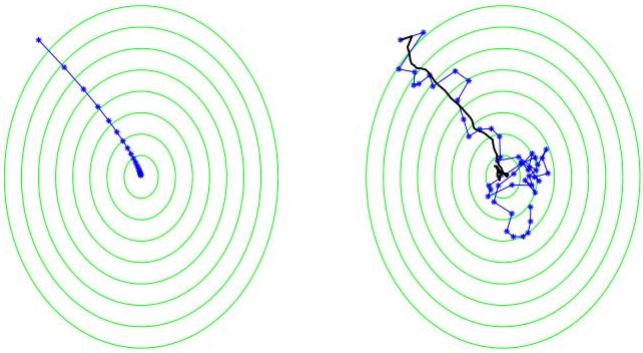


Figure A.18: Gradient descent algorithm (left) and stochastic gradient descent (right) paths. For the stochastic case, the black line depicts the averaged value of $\mathbf{x}$, reprinted from [16] with permission of Cambridge University Press through PLSclear

Another reason for which SGD works well with neural networks is given

by how partial derivatives can be easily computed exploiting the neural network architecture with the *Backpropagation* approach. The computation of the gradient proceeds backward through the network, the gradient of the final layer of weights is computed first and the gradient of the first layer of weights is computed last. Partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer. This backward flow of the error information allows for efficient computation of the gradient at each layer. The mathematical details that show how SGD can be implemented with backpropagation in neural networks can be found in [16]

A.3 Deep Learning for Image Reconstruction

In the literature there are no works where PET images have been reconstructed starting from the data using deep learning, currently only denoisers have been employed with the aim of improving the quality of PET images [23]. A denoiser is a deep neural network that learns the properties of noise affecting a set of images and then, once applied to a noisy image belonging to this set, is able to denoise it thus improving its quality. Denoisers are applied to PET images that have already been reconstructed using, for example, some iterations of the MLEM algorithm.

A possible approach to image reconstruction using deep learning is proposed in [2] for CT data, this method is called *Primal Dual Reconstruction*. The primal dual network architecture is characterized by two branches, the *Dual* one works in the data domain, inputs and outputs are thus sinograms. The second branch is the *Primal* one and works in the image domain, inputs and outputs are thus images. Each branch is made of a series of U-Nets applied one after the other as Figure A.19 shows. The number of networks in a branch corresponds to the number of iterations i of the algorithm.

Each U-Net of the Dual branch has three inputs:

- h_i: denoised sinogram after i iterations,
- $T(f_i)$: sinogram obtained by applying the forward operator T to the reconstructed image after i iterations,
- g: sinogram obtained from measured data.

The output of each dual iteration h_{i+1} is the sum of the input denoised sinogram h_i and the dual U-Net output.

Each U-Net of the Primal branch has two inputs:

- $T^*(h_{i+1})$: image obtained by applying the adjoint of the forward operator T^* to the output of the dual iteration h_{i+1},
- f_i: reconstructed image after i iterations.

The output of each primal iteration f_{i+1} is the sum of the input reconstructed image f_i and the primal U-Net output.

The primal dual network architecture is obtained connecting multiple dual and primal networks one after the other as Figure A.19 shows. The reconstructed image is the output of the last primal iteration.

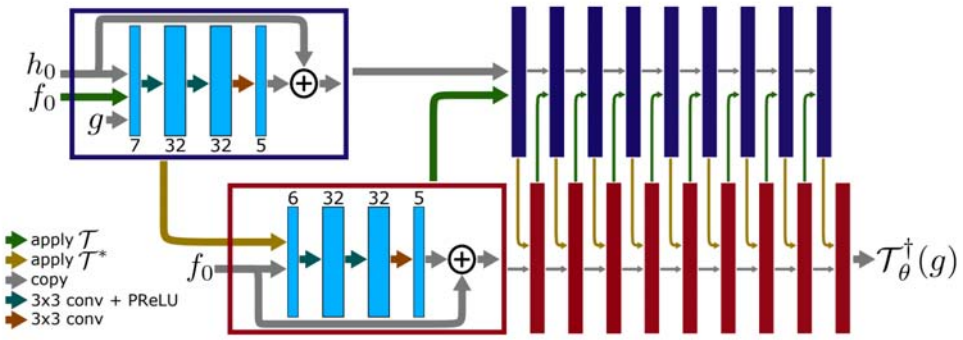


Figure A.19: Primal Dual network architecture, reprinted from [2] use permitted under the Creative Commons Attribution License CC BY

The initial reconstructed image and denoised sinogram, namely f_0 and h_0, can be set equal to 0 or 1 there is no difference in the final result and in convergence time [2].

This approach can be applied also to PET data but there is a key difference that make the PET reconstruction more challenging: PET images have an higher level of noise than CT images, it is thus harder for the network to distinguish between noise and small features that may be present in the object from which data are acquired.

Bibliography

[1] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[2] Jonas Adler and Ozan Öktem. “Learned Primal-Dual Reconstruction”. In: *IEEE Transactions on Medical Imaging* 37.6 (June 2018), pp. 1322–1332.

[3] Timothy G Turkington. “Introduction to PET instrumentation”. In: *Journal of nuclear medicine technology* 29.1 (2001), pp. 4–11.

[4] Frederic H. Fahey. “Data Acquisition in PET Imaging”. In: *Journal of Nuclear Medicine Technology* 30.2 (2002), pp. 39–49.

[5] Sergio DeBenedetti et al. “On the angular distribution of two-photon annihilation radiation”. In: *Physical Review* 77.2 (1950), p. 205.

[6] Kengo Shibuya et al. “Annihilation photon acollinearity in PET: volunteer and phantom FDG studies”. In: *Physics in Medicine & Biology* 52.17 (2007), p. 5249.

[7] William W. Moses. “Fundamental limits of spatial resolution in PET”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 648 (2011), S236–S240. ISSN: 0168-9002.

[8] Michel Defrise and Paul Kinahan. “Data Acquisition and Image Reconstruction for 3D PET”. In: *The Theory and Practice of 3D PET* (Jan. 1, 1998).

[9] Steven W Smith et al. “The scientist and engineer’s guide to digital signal processing”. In: (1997).

[10] Rafael C Gonzalez, Richard E Woods, et al. *Digital image processing*. 2002.

[11] Artur Słomski et al. "3D PET image reconstruction based on the maximum likelihood estimation method (MLEM) algorithm". In: *Bio-Algorithms and Med-Systems* 10.1 (2014), pp. 1–7.

[12] Toshiyuki Yokoi et al. "Implementation and performance evaluation of iterative reconstruction algorithms in SPECT: a simulation study using EGS4". In: *KEK PROCEEDINGS*. High Energy Accelerator Research Organization; 1999. 2000, pp. 224–234.

[13] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

[14] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[15] Hong Sun, Cheng-wei Sang, and Didier Le Ruyet. "Sparse signal subspace decomposition based on adaptive over-complete dictionary". In: *EURASIP Journal on Image and Video Processing* 2017.1 (July 2017), p. 50. ISSN: 1687-5281.

[16] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[17] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X.

[18] Luca Frau and Alessandro Chiuso. "Appunti delle lezioni del corso di machine learning". In: *Unpublished* (2018).

[19] Jumpei Ukita, Takashi Yoshida, and Kenichi Ohki. "Characterisation of nonlinear receptive fields of visual neurons by convolutional neural network". In: *Scientific reports* 9.1 (2019), p. 3791.

[20] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *ArXiv* abs/1505.04597 (2015).

[21] Kyong Hwan Jin et al. "Deep Convolutional Neural Network for Inverse Problems in Imaging". In: *IEEE Transactions on Image Processing* 26.9 (Sept. 2017), pp. 4509–4522. ISSN: 1941-0042.

[22] Zhenhua Feng et al. “Wing Loss for Robust Facial Landmark Localisation with Convolutional Neural Networks”. In: *ArXiv* (June 2018).

[23] Kuang Gong et al. “Pet image denoising using a deep neural network through fine tuning”. In: *IEEE Transactions on Radiation and Plasma Medical Sciences* 3.2 (2018), pp. 153–161.