Pattern Recognition Lab

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Python Reconstruction Operators in Neural Networks

PYRO-NN

PYRO-NN

- Geometries
- Trajectories
- Filters
- Layers

Base-Geometry Properties

Volume

- The volume size in [Z, Y, X] order
- The spacing between voxels in [Z, Y, X] order.

❖ Volume center is world coordinate origin

Detector

- Shape of the detector in [Y, X] order.
- The spacing between detector voxels in [Y, X] order.

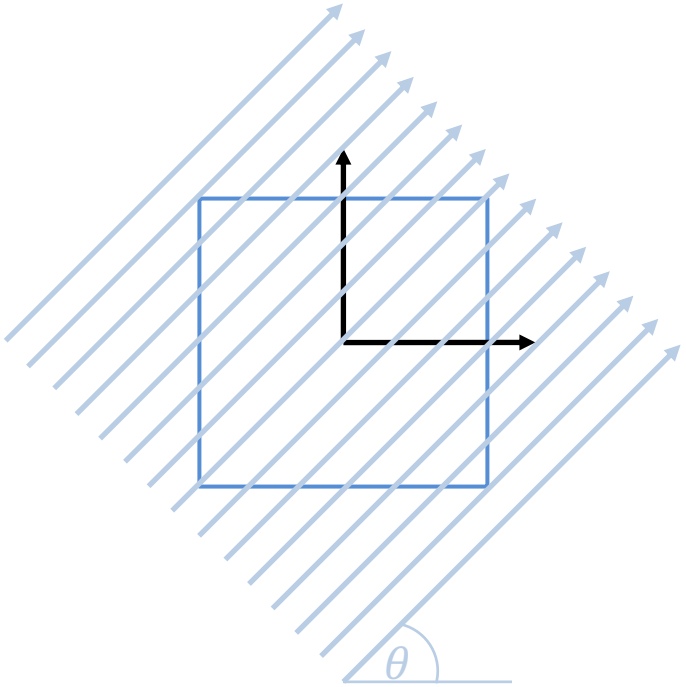
Acquisition Geometry

- Number of equidistant projections.
- The covered angular range.

2D Parallel-Geometry Properties

Trajectory

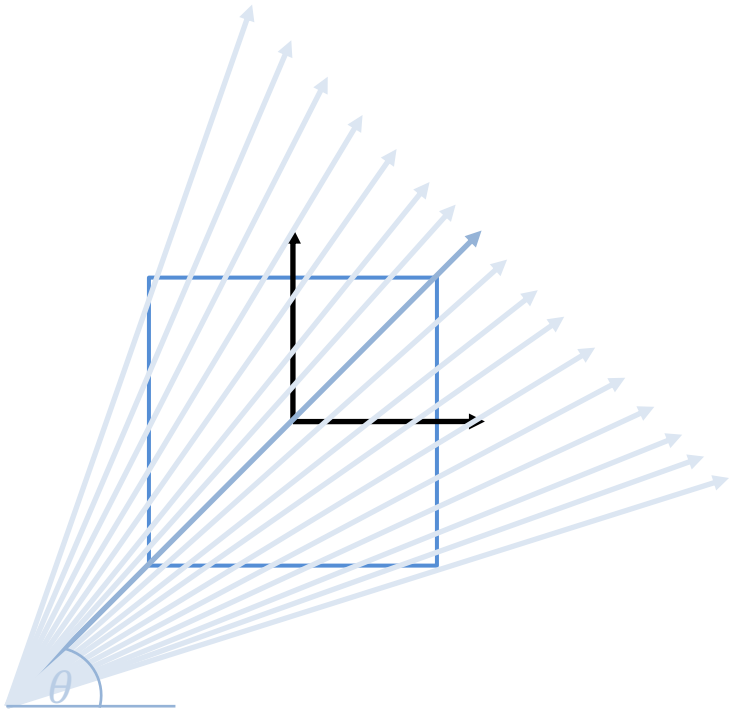
- based on ray vectors
- unit vectors with angle $\theta \in$ [0 ; angular range]



2D Fan-Geometry Properties

Trajectory

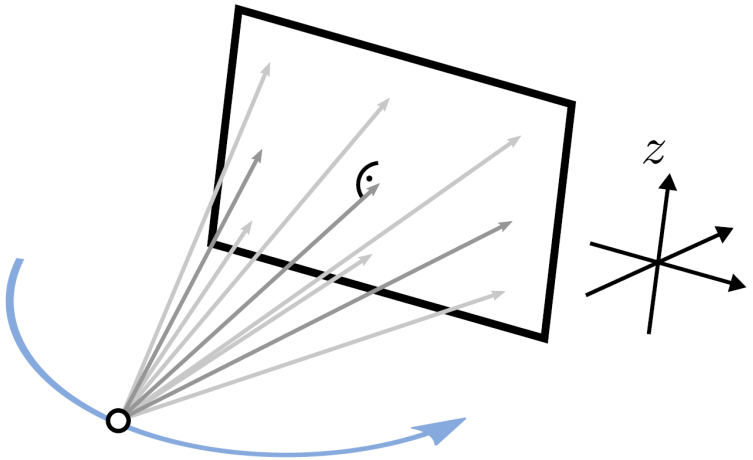
- based on central-ray vectors of the fan-beam
- unit vectors with angle $\theta \in [0 ;$ angular range$]$
- Source to isocenter distance (sid)
- Source to detector distance (sdd)



3D Cone-Geometry Properties

Trajectory

- based on projection matrices
  - ❖ Calibrated matrices from real systems can be used
- Source to isocenter distance (sid)
- Source to detector distance (sdd)



Maier, Andreas, et al., eds. *Medical Imaging Systems: An Introductory Guide*. Vol. 11111. Springer, 2018.

2D & 3D Trajectory Generator

Circular Trajectory 2D:

- Derived from defined geometry (2D parallel / fan-beam)
- Computes vector array

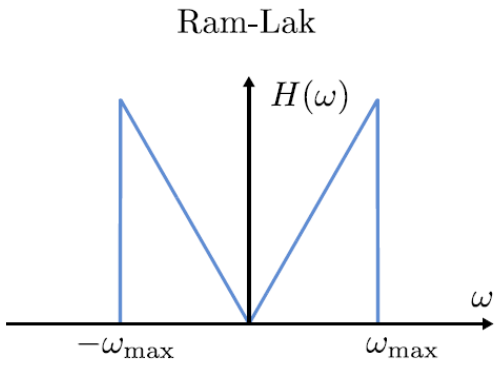
Circular Trajectory 3D:

- Projection matrices based on defined 3D cone-beam geometry
- Computes projection matrices

Filters

Reconstruction Filters

- Ramp filter for 2D & 3D
- Ram-Lak filter for 2D & 3D



Maier, Andreas, et al., eds. *Medical Imaging Systems: An Introductory Guide*. Vol. 11111. Springer, 2018.

Geometric Correction Weights

- Cosine-weighting (2D fan & 3D cone)
- Parker-weighting for short-scans (2D fan & 3D cone)

Layers

Forward projector

- 2D parallel- and fan-beam
- 3D cone-beam

❖ Implemented as ray-driven

Backprojector

- 2D parallel- and fan-beam
- 3D cone-beam

❖ Implemented as voxel-driven

❖ **Gradient are internally already registered**

PYRO-NN Examples

- Examples are provided within the PYRO-NN Repository under https://github.com/csyben/PYRO-NN

- Following examples are provided in a runnable code capsule under https://codeocean.com/capsule/6772846/

Example: 2D Parallel

Geometry & Phantom definitions

```
#Imports
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from pyronn.ct_reconstruction.layers.projection_2d import parallel_projection2d
from pyronn.ct_reconstruction.layers.backprojection_2d import
parallel_backprojection2d
from pyronn.ct_reconstruction.geometry.geometry_parallel_2d import
GeometryParallel2D
from pyronn.ct_reconstruction.helpers.filters import filters
from pyronn.ct_reconstruction.helpers.phantoms import shepp_logan
from pyronn.ct_reconstruction.helpers.trajectories import circular_trajectory
```

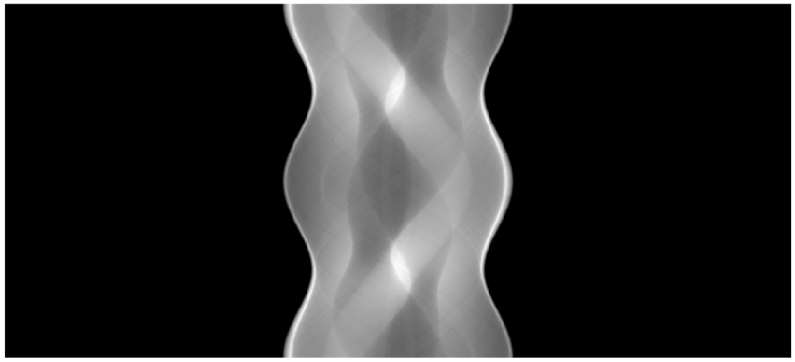
```
# ------------------ Define Geometry------------------
    # Volume Parameters:
    volume_size = 256
    volume_shape = [volume_size, volume_size]
    volume_spacing = [1, 1]
    # Detector Parameters:
    detector_shape = 800
    detector_spacing = 1
    # Trajectory Parameters:
    number_of_projections = 360
    angular_range = 2* np.pi
    # create Geometry class
    geometry = GeometryParallel2D(volume_shape, volume_spacing,
                                  detector_shape, detector_spacing,
                                  number_of_projections, angular_range)
    #compute and set trajectory
    trajectory_vectors = circular_trajectory.circular_trajectory_2d(geometry))
    geometry.set_ray_vectors(trajectory_vectors )

    # Get Phantom
    phantom = shepp_logan.shepp_logan_enhanced(volume_shape)

    phantom = np.expand_dims(phantom, axis=0) #Add batch dimension
```

Example: 2D Parallel

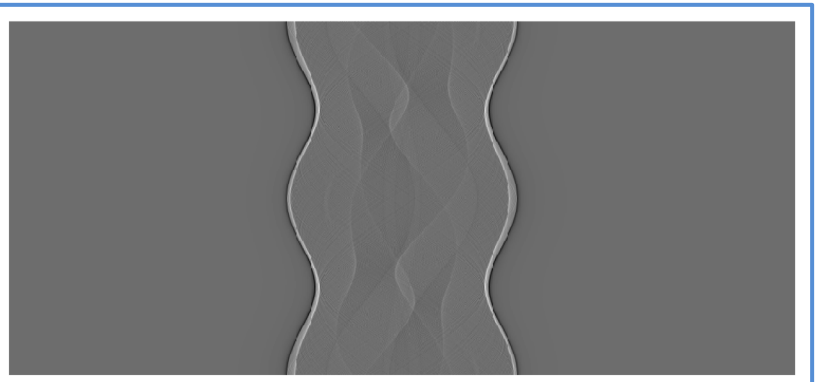
Projection & Reconstruction

```
#Imports
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from pyronn.ct_reconstruction.layers.projection_2d import parallel_projection2d
from pyronn.ct_reconstruction.layers.backprojection_2d import
parallel_backprojection2d
from pyronn.ct_reconstruction.geometry.geometry_parallel_2d import
GeometryParallel2D
from pyronn.ct_reconstruction.helpers.filters import filters
from pyronn.ct_reconstruction.helpers.phantoms import shepp_logan
from pyronn.ct_reconstruction.helpers.trajectories import circular_trajectory
```

```
# ------------------ Call Layers ------------------
    with tf.Session() as sess:
        #Create sinogram according to geometry
        result = parallel_projection2d(phantom, geometry)
        sinogram = result.eval()
```



```
        # Get filter
        reco_filter = filters.ram_lak_2D(geometry)

        # Filter sinogram in frequency domain
        sino_freq = np.fft.fft(sinogram, axis=-1)
        sino_filtered_freq = np.multiply(sino_freq,reco_filter)
        sinogram_filtered = np.fft.ifft(sino_filtered_freq, axis=-1).real
```

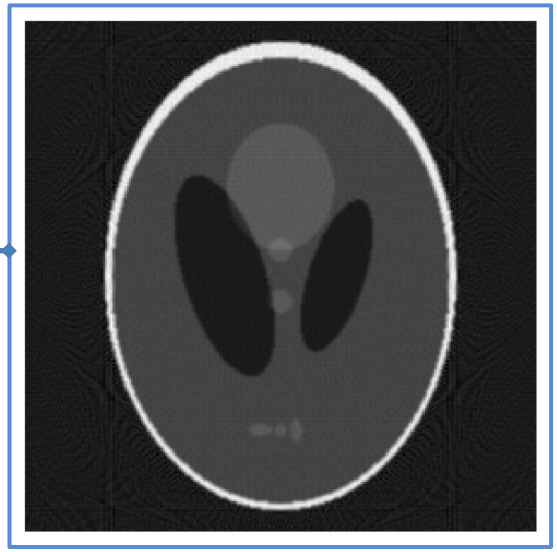


Example: 2D Parallel

Projection & Reconstruction

```
#Imports
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from pyronn.ct_reconstruction.layers.projection_2d import parallel_projection2d
from pyronn.ct_reconstruction.layers.backprojection_2d import
parallel_backprojection2d
from pyronn.ct_reconstruction.geometry.geometry_parallel_2d import
GeometryParallel2D
from pyronn.ct_reconstruction.helpers.filters import filters
from pyronn.ct_reconstruction.helpers.phantoms import shepp_logan
from pyronn.ct_reconstruction.helpers.trajectories import circular_trajectory
```

```
# ------------------ Call Layers ------------------
    with tf.Session() as sess:

        .....

    # Reconstruction from filtered sinogram
        result_back_proj = parallel_backprojection2d(sinogram_filtered, geometry)
        reco = result_back_proj.eval()
```



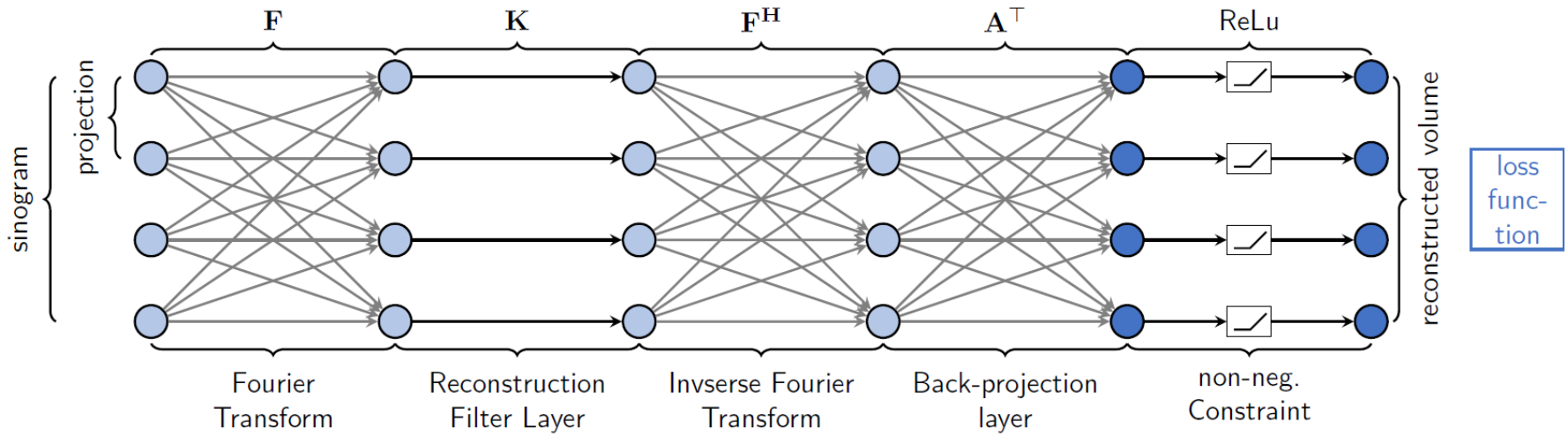
```
    #Visualize result
    plt.figure()
    plt.imshow(reco, cmap=plt.get_cmap('gist_gray'))
    plt.axis('off')
    plt.savefig('2d_par_reco.png', dpi=150, transparent=False, bbox_inches='tight')
```

Example: Learn Filter

Situation: Ramp filter is correct for continuous systems

➢ Discretization error leads to offset & cupping artifacts

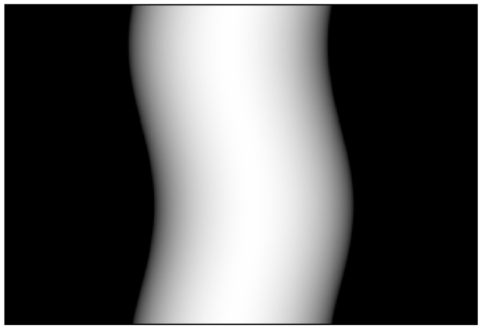
Goal: Learn the correct discretization for the reconstruction filter [1]



[1] C. Syben et al. Precision learning: Reconstruction filter kernel discretization, in Proceedings of the Fifth International Conference on Image Formation in X-Ray Computed Tomography, pages 386–390, 2018

Recap: CT Reconstruction

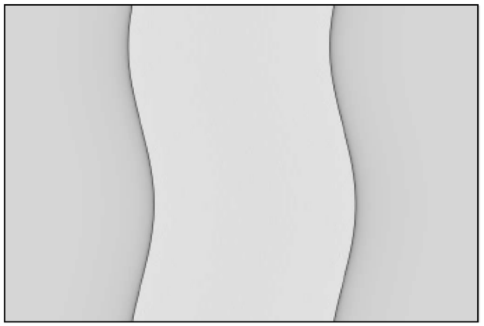
Sinogram



$p(s, \theta)$

$\xrightarrow{h(s) * p(s, \theta)}$

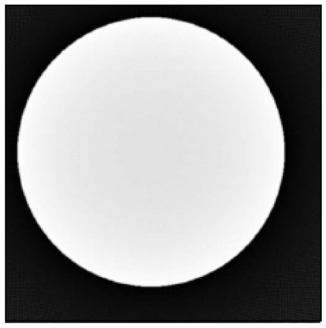
Filtered sinogram



$q(s, \theta)$

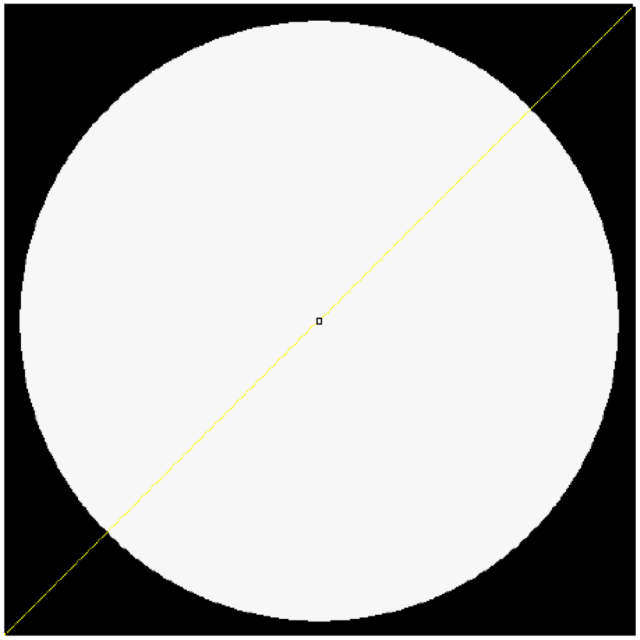
$\xrightarrow{f(x, y) = \int q(s, \theta) d\theta}$

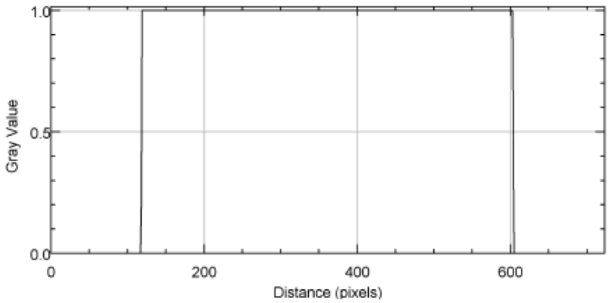
Reconstruction



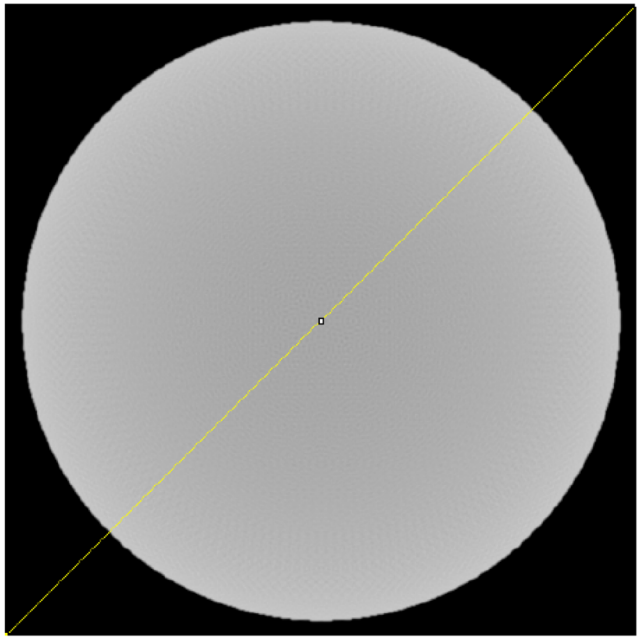
where $h(s)$ is the Ramp-Filter

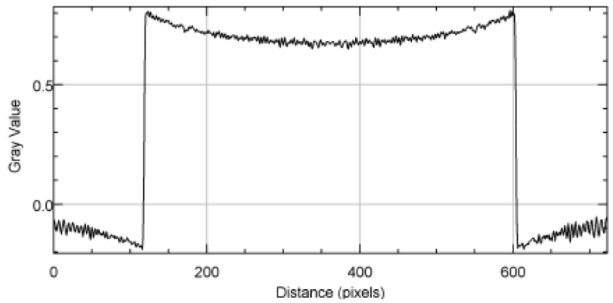
Cupping Artifacts





Line profile through phantom.





Line profile through Ramp-Reco

Deriving the Network Topology

Discrete reconstruction problem:

$$\mathbf{A}\mathbf{x} = \mathbf{p}$$

$$\mathbf{x} = \underbrace{\mathbf{A}^\top}_{\text{Back-projection}} \underbrace{\left(\mathbf{A}\mathbf{A}^\top\right)^{-1}}_{\text{Filter}} \mathbf{p}$$

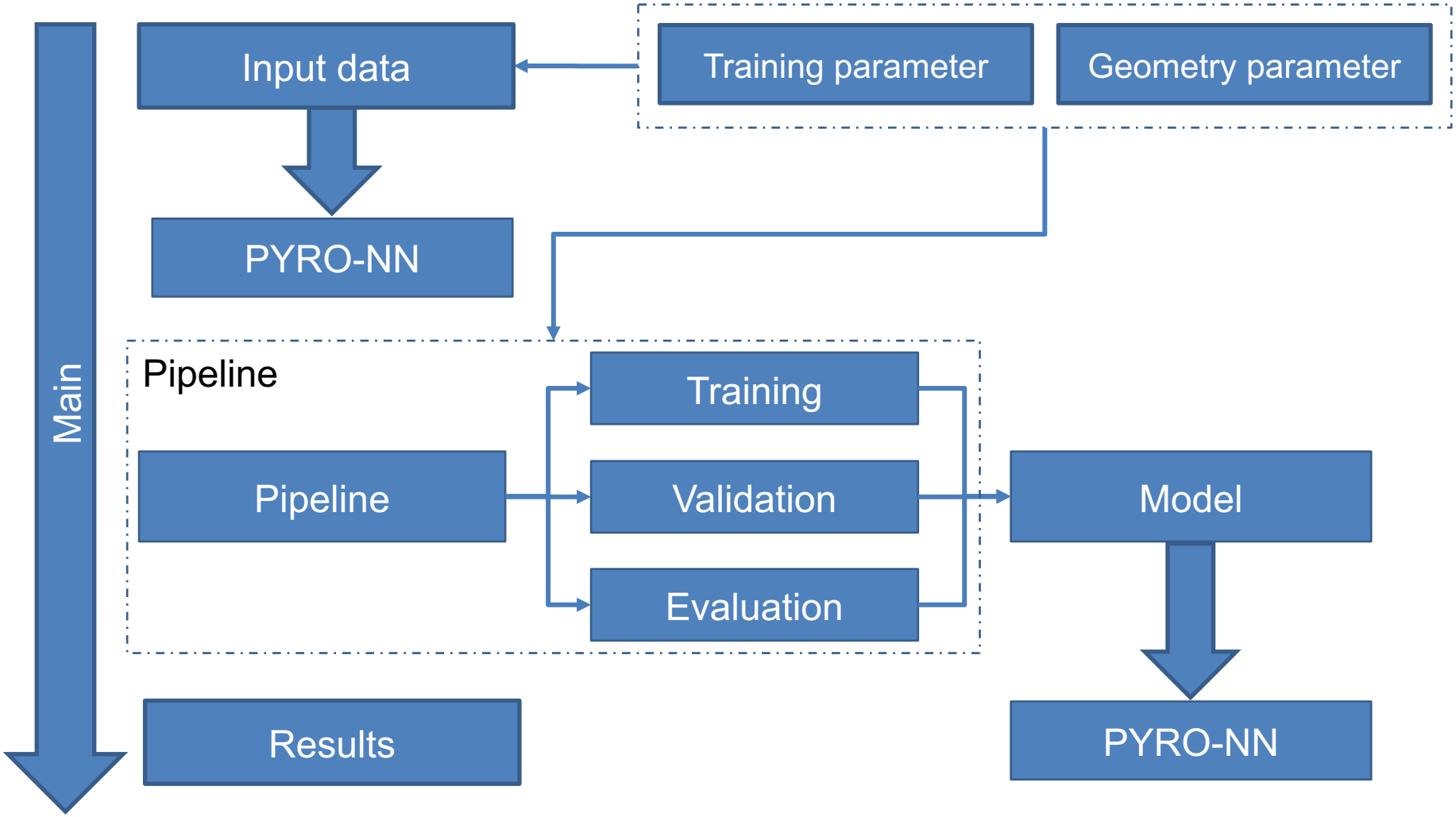
substituting the inverse:

$$\mathbf{x} = \mathbf{A}^\top \mathbf{F}^\mathbf{H} \mathbf{K} \mathbf{F} \mathbf{p}$$

where

- $\mathbf{A}$ is the system matrix
- $\mathbf{x}$ is the object
- $\mathbf{p}$ is the sinogram
- $\mathbf{F}, \mathbf{F}^\mathbf{H}$ is the Fourier and inverse Fourier-transform
- $\mathbf{K}$ is the filter in Fourier domain

Example: Learn Reco Filter

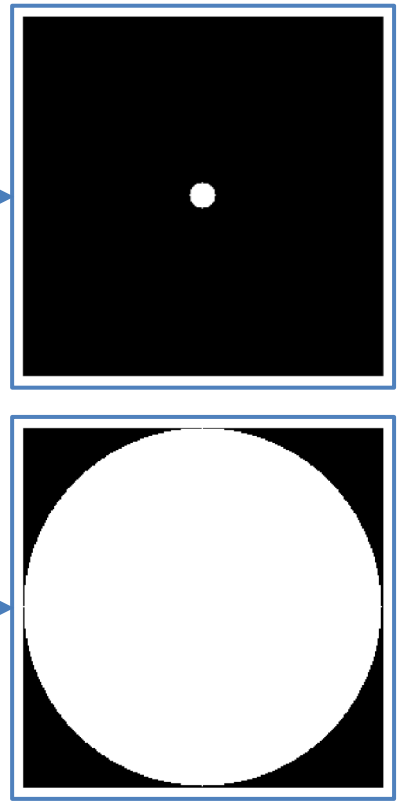


Learn Reco Filter

Data generation I (input_data.py)

```
#Imports
import numpy as np
import tensorflow as tf
from model.geometry_parameter import GEOMETRY

#PYRO-NN Phantoms
from pyronn.ct_reconstruction.helpers.phantoms import primitives_2d,
shepp_logan
from pyronn.ct_reconstruction.layers.projection_2d import parallel_projection2d
from pyronn.ct_reconstruction.helpers.misc.generate_sinogram import
generate_sinogram
from pyronn.ct_reconstruction.helpers.misc.generate_sinogram import
generate_sinogram_parallel_2d
```

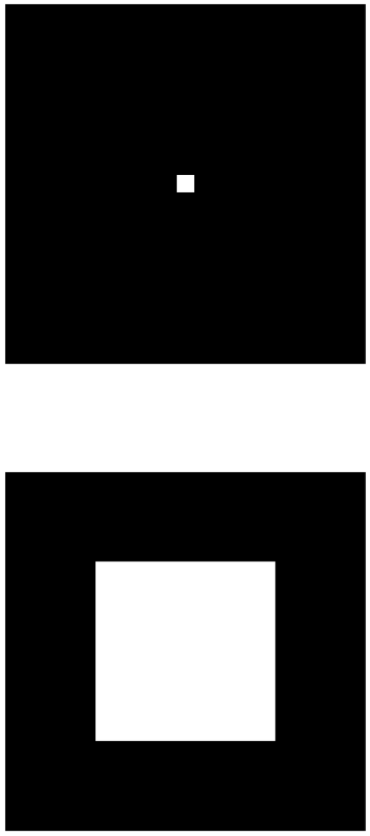
```
#Training data := Circles with increasing radii (+2 pixel)
def generate_training_data():
    label_list = []
    input_data_list = []
    max_radius = np.min(GEOMETRY.volume_shape) // 2
    center_pos = [(GEOMETRY.volume_shape[0]-1)//2, (GEOMETRY.volume_shape[1]-1)//2]
    #Compute phantom
    for n in range(9, max_radius, 2):
        #Add batch dimension with dim == 1 for sinogram generation
        phantom = primitives_2d.circle(GEOMETRY.volume_shape, center_pos, n)
        label_list.append(np.expand_dims( phantom, axis=0))
    #Create sinogram data
    with tf.Session() as sess:
        for phantom in label_list:
            sinogram = generate_sinogram_parallel_2d(phantom, GEOMETRY)
            input_data_list.append(sinogram)
    #Remove batch dimension | Return input & label
    return np.squeeze(np.asarray(input_data_list)), np.squeeze(np.asarray(label_list))
```



Learn Reco Filter

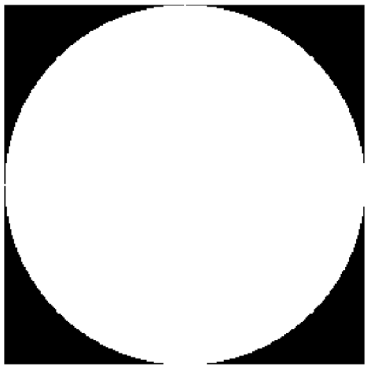
Data generation II (input_data.py)

Validation



Test

Cupping Test

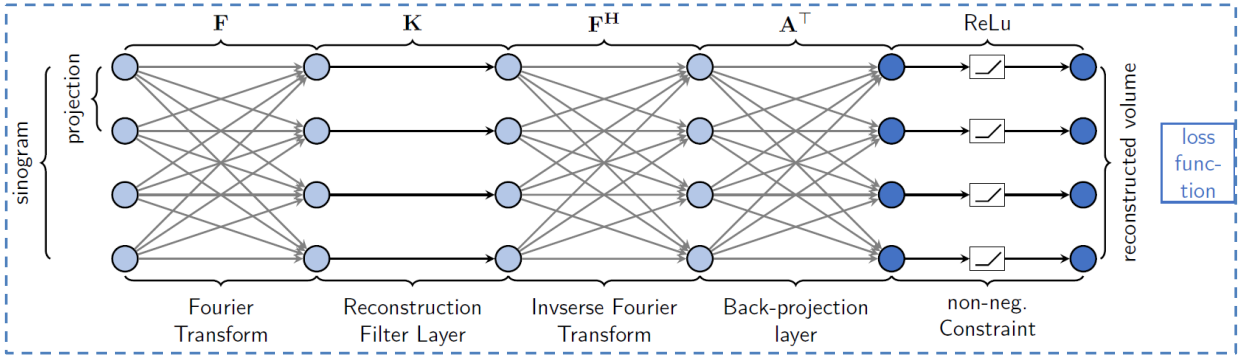


Learn Reco Filter

Model

```
#Imports
import tensorflow as tf
from model.geometry_parameter import GEOMETRY

#PYRO-NN
from pyronn.ct_reconstruction.layers.backprojection_2d import
parallel_backprojection2d
from pyronn.ct_reconstruction.helpers.filters.filters import ramp
```

```
class filter_model:
    def __init__(self):
        filter = ramp(GEOMETRY.detector_shape[0])
        # Define filter weights tensor. Initialize with ramp filter
        self.filter_weights = tf.get_variable(name='filter_frequency', dtype=tf.float32, initializer=filter, trainable=True)
        self.filter_weights_placeholder = tf.placeholder(tf.float32, name='filter_weights_placeholder')
        self.set_filter_weights = self.filter_weights.assign(self.filter_weights_placeholder)

    def forward(self, input_sinogram):
        sinogram_frequency = tf.fft(tf.cast(input_sinogram,dtype=tf.complex64))
        # Filtering step.
        filtered_sinogram_frequency = tf.multiply(sinogram_frequency, tf.cast(self.filter_weights,dtype=tf.complex64))
        filtered_sinogram = tf.real(tf.ifft(filtered_sinogram_frequency))
        # Reconstrution using PYRO-NN parallel backprojection layer
        reco = parallel_backprojection2d(filtered_sinogram, GEOMETRY)
        return reco, self.filter_weights
```



Learn Reco Filter

Pipeline I

```
#Imports
import tensorflow as tf
import os

from model import training_parameter as args
from model.geometry_parameter import GEOMETRY
from model.model import filter_model
```

```
class pipeline:
    """
        Here the training environment in term of Tensorflow is defined, including all necessary structure for training and
        validation datasets, loss computation and optimization.
    """
    def __init__(self, session):
        self.sess    = session                  # The Tensorflow session is made available to the pipeline methods
        self.model  = filter_model()            # The defined model is used
        self.results = dict()

    def init_placeholder_graph(self):
        # define placeholder for the learning process, including learning rate, average loss and train/test switch
        self.learning_rate = tf.get_variable(name='learning_rate', dtype=tf.float32, initializer=tf.constant(0.0001))
        self.learning_rate_placeholder = tf.placeholder(tf.float32, name='learning_rate_placeholder')
        self.set_learning_rate = self.learning_rate.assign(self.learning_rate_placeholder)

        self.is_training = tf.get_variable(name="is_training", shape=[], dtype=tf.bool, trainable=False)
        self.set_training = self.is_training.assign(True)
        self.set_validation = self.is_training.assign(False)

        self.avg_loss_placeholder = tf.placeholder(tf.float32, name='avg_loss_placeholder')
        self.avg_validation_loss_placeholder = tf.placeholder(tf.float32, name='avg_validation_loss_placeholder')
```

Learn Reco Filter

Pipeline II

```
#Imports
import tensorflow as tf
import os

from model import training_parameter as args
from model.geometry_parameter import GEOMETRY
from model.model import filter_model
```

```
class pipeline:
…
    """
        Here the training environment in term of Tensorflow is defined, including all necessary structure for training and
        validation datasets, loss computation and optimization.
    """
  def data_loader(self, inputs, labels):
    # Data iterator
    # Make pairs of elements. (X, Y) => ((x0, y0), (x1)(y1)),....
    image_set = tf.data.Dataset.from_tensor_slices((inputs, labels))
    # Identity mapping operation is needed to include multi-tthreaded queue buffering.
    image_set = image_set.map(lambda x, y: (x, y), num_parallel_calls=4).prefetch(buffer_size=200)
    # Batch dataset. Also do this if batchsize==1 to add the mandatory first axis for the batch_size
    image_set = image_set.batch(1)
    # Repeat dataset for number of epochs
    image_set = image_set.repeat(args.MAX_EPOCHS+1)
    # Prefetch data to gpu.
    # Select iterator
    iterator = image_set.make_initializable_iterator()
    return iterator
```

Learn Reco Filter

Pipeline III

```
#Imports
import tensorflow as tf
import os

from model import training_parameter as args
from model.geometry_parameter import GEOMETRY
from model.model import filter_model
```

```
class pipeline:
…
    def build_graph(self):
        self.init_placeholder_graph()                                   # Set Placeholders
        optimizer = tf.train.AdamOptimizer(self.learning_rate, epsilon=0.1)   # Optimizer

        # Tensor placeholders that are initialized later. Placeholder for training and test dataset
        self.inputs_train = tf.placeholder(tf.float32, (None, *GEOMETRY.sinogram_shape))
        self.labels_train = tf.placeholder(tf.float32, (None, *GEOMETRY.volume_shape))
        self.inputs_validation = tf.placeholder(tf.float32, (None, *GEOMETRY.sinogram_shape))
        self.labels_validation = tf.placeholder(tf.float32, (None, *GEOMETRY.volume_shape))
        # Get next_element-"operator" and iterator that is initialized later
        self.iterator_train = self.data_loader(self.inputs_train, self.labels_train)
        self.iterator_validation = self.data_loader(self.inputs_validation, self.labels_validation)
        # Get next (batch of) element pair(s)
        self.input_element, self.label_element = tf.cond(self.is_training,
                                                         lambda: self.iterator_train.get_next(),
                                                         lambda: self.iterator_validation.get_next())
        # Model and loss function
        self.prediction, self.filter_weights = self.model.forward(self.input_element)  # Model evaluation
        self.loss = self.model.l2_loss(self.prediction, self.label_element)             # Compute loss
        self.train_op = optimizer.minimize(self.loss)                                   # Update weights

        # Summary stuff
        …
```

Learn Reco Filter

Pipeline IV

```
#Imports
import tensorflow as tf
import os

from model import training_parameter as args
from model.geometry_parameter import GEOMETRY
from model.model import filter_model
```

```
class pipeline:
…
    def validation(self, epoch):
        # Switch to validation dataset
        self.sess.run(self.set_validation)
        avg_validation_loss = 0
        for step in range(0, args.NUM_VALIDATION_SAMPLES):
            # Do one step of model validation (no weight update)
            validation_loss, reco, current_filter = self.sess.run([self.loss, self.prediction, self.filter_weights])
            avg_validation_loss=+validation_loss
        # Compute average validation error
        self.avg_validation_loss = avg_validation_loss / args.NUM_VALIDATION_SAMPLES
        print("epoch: %d | avg validation loss: %f" % (epoch, self.avg_validation_loss))
        # Switch back to training dataset
        self.sess.run(self.set_training)
```

Learn Reco Filter

Pipeline V

```
#Imports
import tensorflow as tf
import os

from model import training_parameter as args
from model.geometry_parameter import GEOMETRY
from model.model import filter_model
```

```
class pipeline:
…
    def train(self, inputs_train, labels_train, inputs_validation, labels_validation):
        #Setup & initialize graph
        self.build_graph()
        self.sess.run(tf.global_variables_initializer())
        self.sess.run(tf.local_variables_initializer())
        #Saver
        self.saver = tf.train.Saver(max_to_keep=100)
        # Feed training (input, label) and validation (input,label) to the previously defined data_loader
        self.sess.run(self.iterator_train.initializer,
                                    feed_dict={self.inputs_train: inputs_train, self.labels_train: labels_train})
        self.sess.run(self.iterator_validation.initializer,
                                    feed_dict={self.inputs_validation: inputs_validation, self.labels_validation: labels_validation})
        # Set training mode & learning rate
        _ = self.sess.run([self.set_training, self.set_learning_rate],
                                    feed_dict={self.learning_rate_placeholder: args.LEARNING_RATE})
        # Save initial filter for result generation
        self.results.setdefault('initial_filter', self.model.get_filter(self.sess))
        print("Start Training")
        loss_epoch_before = 1e16

        … # Training next slide
```

Learn Reco Filter

Pipeline VI

```
#Imports
import tensorflow as tf
import os

from model import training_parameter as args
from model.geometry_parameter import GEOMETRY
from model.model import filter_model
```

```
class pipeline:
…
    def train(self, inputs_train, labels_train, inputs_validation, labels_validation):
        …
        for epoch in range(1,args.MAX_EPOCHS+1):
            avg_loss = 0
            for step in range(0, len(inputs_train)):
                # Do one step of model evaluation and subsequent weight update with train_op
                _, loss, reco, current_filter = self.sess.run([self.train_op,self.loss,self.prediction, self.filter_weights])
                avg_loss += loss
            # Compute average loss over epoch
            avg_loss = avg_loss / len(inputs_train)
            print("epoch: %d | avg epoch loss: %f"%(epoch, avg_loss ))
            # Do one validation step
            self.validation(epoch)
            # Early stopping if loss is increasing or staying the same after one epoch
            if avg_loss >= loss_epoch_before:
                # Save best model
                self.saver.save(self.sess,args.WEIGHTS_DIR, global_step=epoch * args.MAX_TRAIN_STEPS)
                break

            loss_epoch_before = avg_loss
        print("training finshed")
        self.results.setdefault('learned_filter', self.model.get_filter(self.sess))          # Save learned filter in result dict
```

Learn Reco Filter

Pipeline VI

```
#Imports
import tensorflow as tf
import os

from model import training_parameter as args
from model.geometry_parameter import GEOMETRY
from model.model import filter_model
```

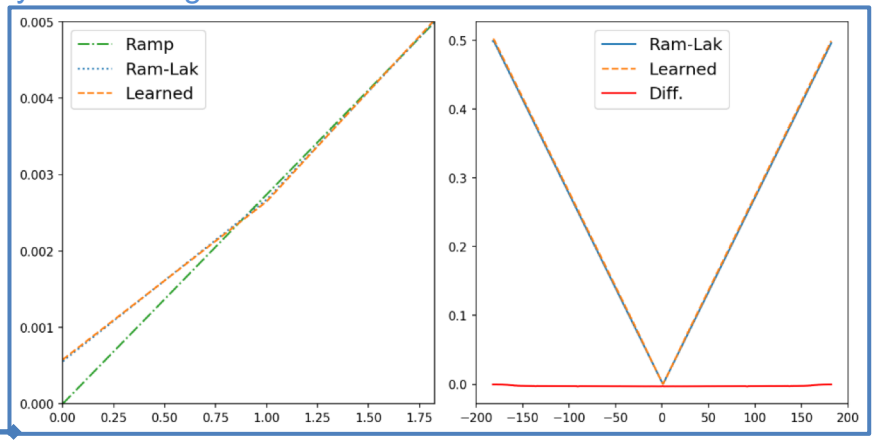
```
class pipeline:
…
    # One pure forward pass of the defined model
    def forward(self, input_data, label_data, filter=None):
        # Switch to validation placeholder
        self.sess.run(self.set_validation)
        # Set filter weights if not None
        if filter is not None:
            self.model.set_filter(self.sess,filter)
        # Feed dataset (input,label) to validation iterator
        self.sess.run(self.iterator_validation.initializer,
                                    feed_dict={self.inputs_validation: input_data, self.labels_validation: label_data})
        avg_loss = 0
        result = []
        for step in range(0, len(input_data)):
            # Do one model evaluation
            loss, reco, current_filter = self.sess.run([self.loss, self.prediction, self.filter_weights])
            avg_loss += loss
            result.append(reco)

        avg_loss/= len(input_data)

        return result, avg_loss
```

Learn Reco Filter

Main

```
import tensorflow as tf
from model import training_parameter as args
from model.geometry_parameter import GEOMETRY
from model.input_data import generate_training_data, generate_validation_data,
get_test_data, get_test_cupping_data
from model.pipeline import pipeline
from pyronn.ct_reconstruction.helpers.filters.filters import ram_lak
from plots import evaluation
```

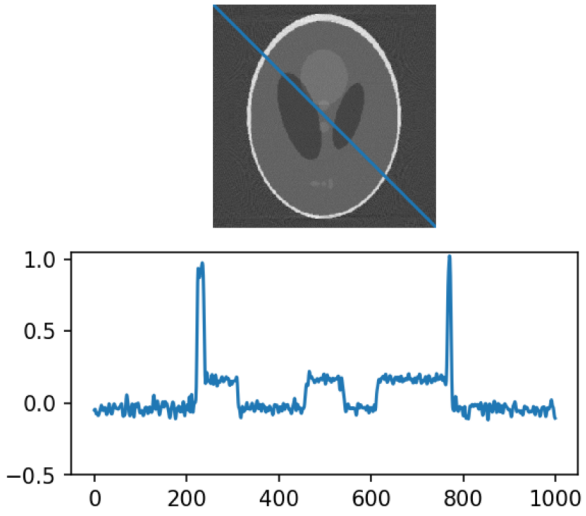
```
# Generate training, validation, test and cupping datasets (input , label)
data, label = generate_training_data()
data_val, label_val = generate_validation_data(args.NUM_VALIDATION_SAMPLES)
data_test, label_test = get_test_data(args.NUM_TEST_SAMPLES)
data_cupping, label_cupping = get_test_cupping_data()

# Configure Tensorflow session using initially 50% of GPU memory and allow growth
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.allow_growth = True
with tf.Session(config=config) as sess:
    # Initialize training pipeline
    training = pipeline(sess)
    # Start training
    training.train(data,label,data_val,label_val)
    # Get Ramp, Ram-Lak and learned filter weights
    initial_filter = training.results.get('initial_filter')
    learned_filter = training.results.get('learned_filter')
    ram_lak_filter = ram_lak(GEOMETRY.detector_shape, GEOMETRY.detector_spacing)

    # Evaluate model with test dataset for Ramp, Ram-Lak and the learned filter
    result_test_initial, rti_avg_loss = training.forward(data_test, label_test, initial_filter)
    result_test_ram_lak, rtrl_avg_loss = training.forward(data_test, label_test, ram_lak_filter)
    result_test_learned, rtl_avg_loss = training.forward(data_test, label_test, learned_filter)
```



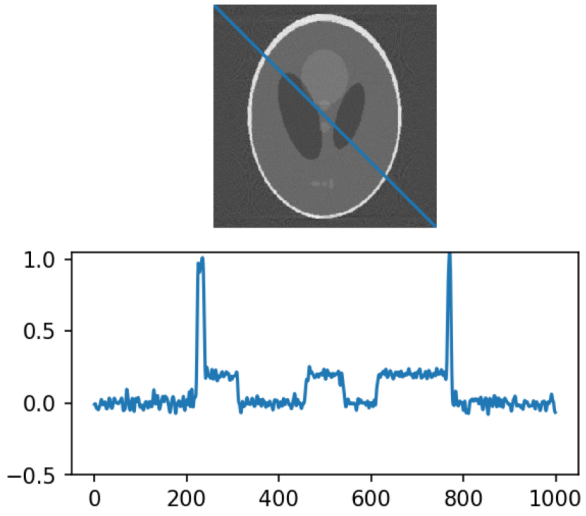
Learn Reco Filter

Results – Test dataset – Shepp-Logan phantom

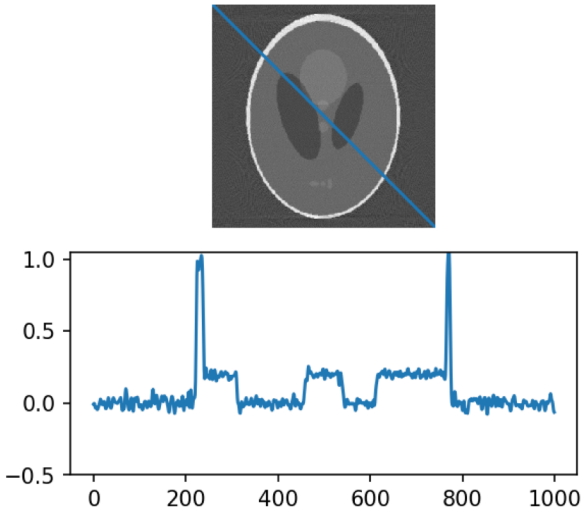
Ramp-Reco



Ram-Lak-Reco



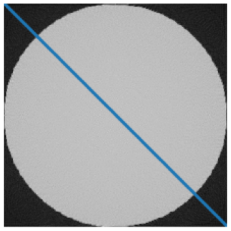
Learned-Reco

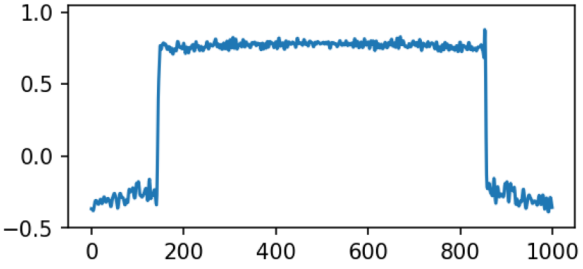


Learn Reco Filter

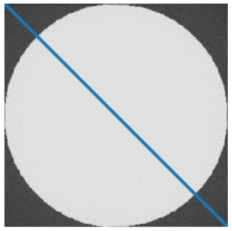
Results – Cupping test dataset

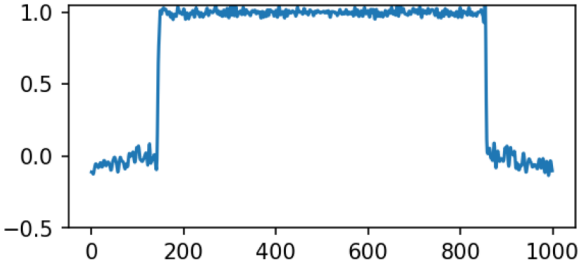
Ramp-Reco



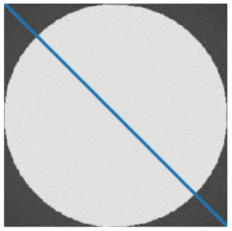


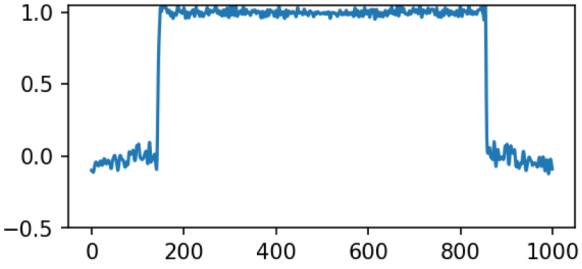
Ram-Lak-Reco





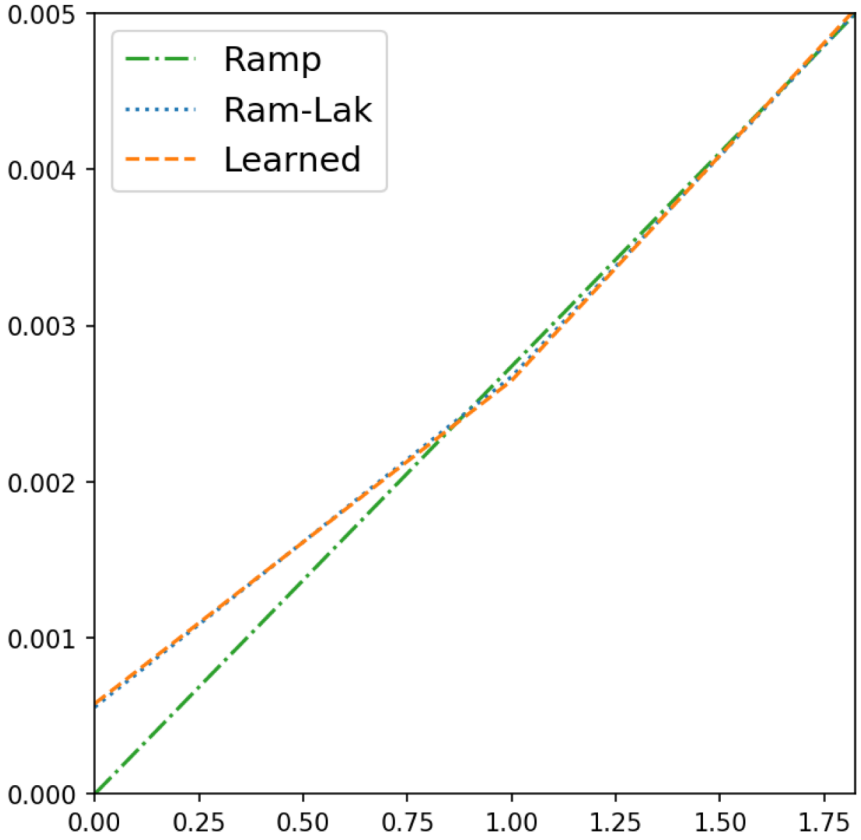
Learned-Reco

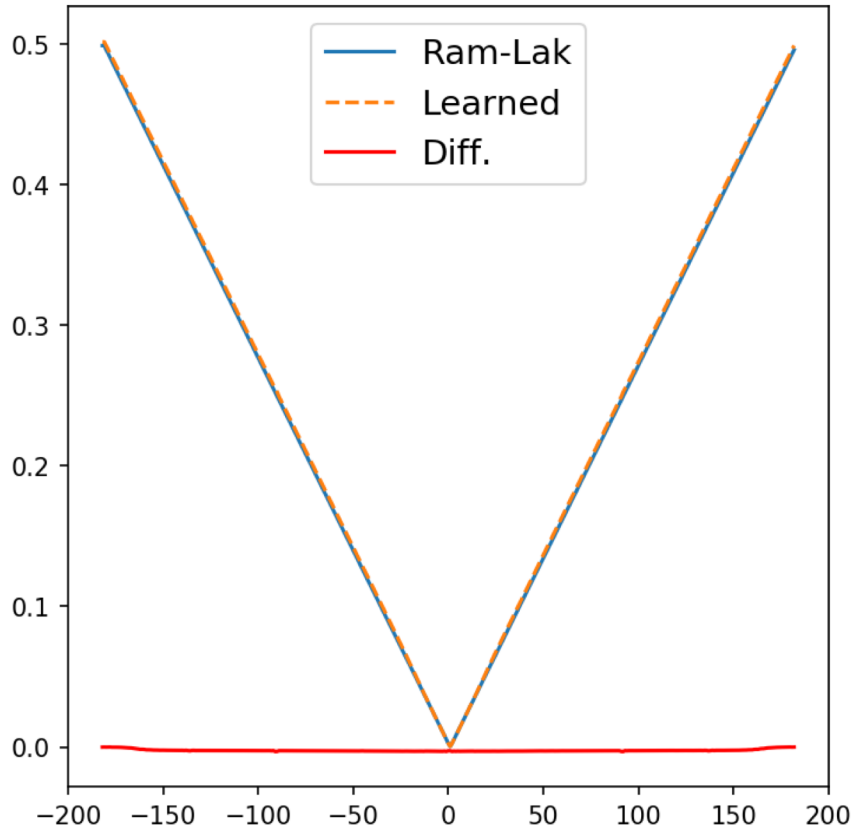




Learn Reco Filter

Results – Learned Filter





Learn Reco Filter

Training parameters

```
#Imports
import tensorflow as tf
from model.geometry_parameter import GEOMETRY

#PYRO-NN
from pyronn.ct_reconstruction.layers.backprojection_2d import
parallel_backprojection2d
from pyronn.ct_reconstruction.helpers.filters.filters import ramp
```

```
# training parameters
LEARNING_RATE                = 1e-6
BATCH_SIZE_TRAIN             = 1
NUM_TRAINING_SAMPLES         = 10
MAX_TRAIN_STEPS              = NUM_TRAINING_SAMPLES//BATCH_SIZE_TRAIN +1
BATCH_SIZE_VALIDATION        = 1
NUM_VALIDATION_SAMPLES       = 10
MAX_VALIDATION_STEPS         = NUM_VALIDATION_SAMPLES//BATCH_SIZE_VALIDATION
NUM_TEST_SAMPLES             = 1
MAX_TEST_STEPS               = NUM_TEST_SAMPLES
MAX_EPOCHS                   = 100
#Path
LOG_DIR = 'logs/'
WEIGHTS_DIR = 'trained_models/'
```

Learn Reco Filter

Geometry parameters

```
#Imports
import numpy as np

#PYRO-NN
from pyronn.ct_reconstruction.geometry_parallel_2d import GeometryParallel2D
from pyronn.ct_reconstruction.helpers.trajectories import circular_trajectory
```

```
"""
    This file defines the Geometry parameters used by the whole model.
    A GeometryParallel2D instance is provided to be used by everyone that needs it.
"""

# Declare Parameters
volume_shape            = [256, 256]
volume_spacing          = [1 , 1]
detector_shape          = 365
detector_spacing        = 1
number_of_projections   = 180
angular_range           = np.pi

# Create Geometry class instance
GEOMETRY = GeometryParallel2D(volume_shape, volume_spacing, detector_shape, detector_spacing,
number_of_projections, angular_range)
# Compute trajectory and set ray vectors
GEOMETRY.set_ray_vectors(circular_trajectory.circular_trajectory_2d(GEOMETRY))
```

Play with the code capsule!

https://codeocean.com/capsule/6772846