Pattern Recognition Lab

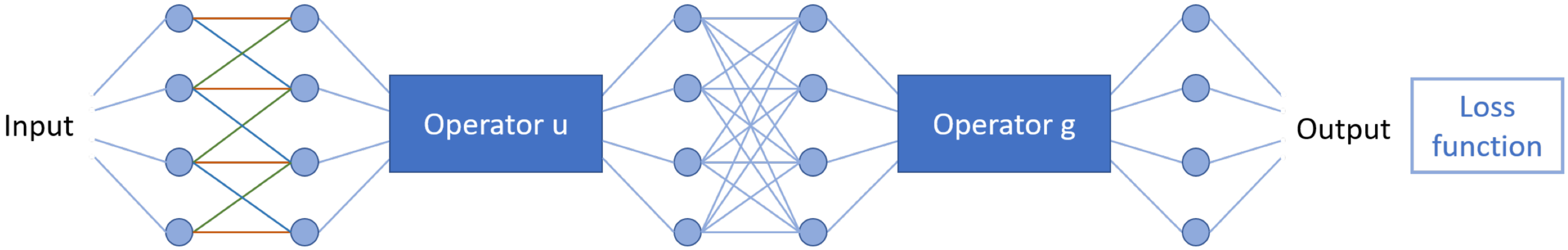
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Python Reconstruction Operators in Neural Networks

PYRO-NN

Motivation

- Use of known operators !

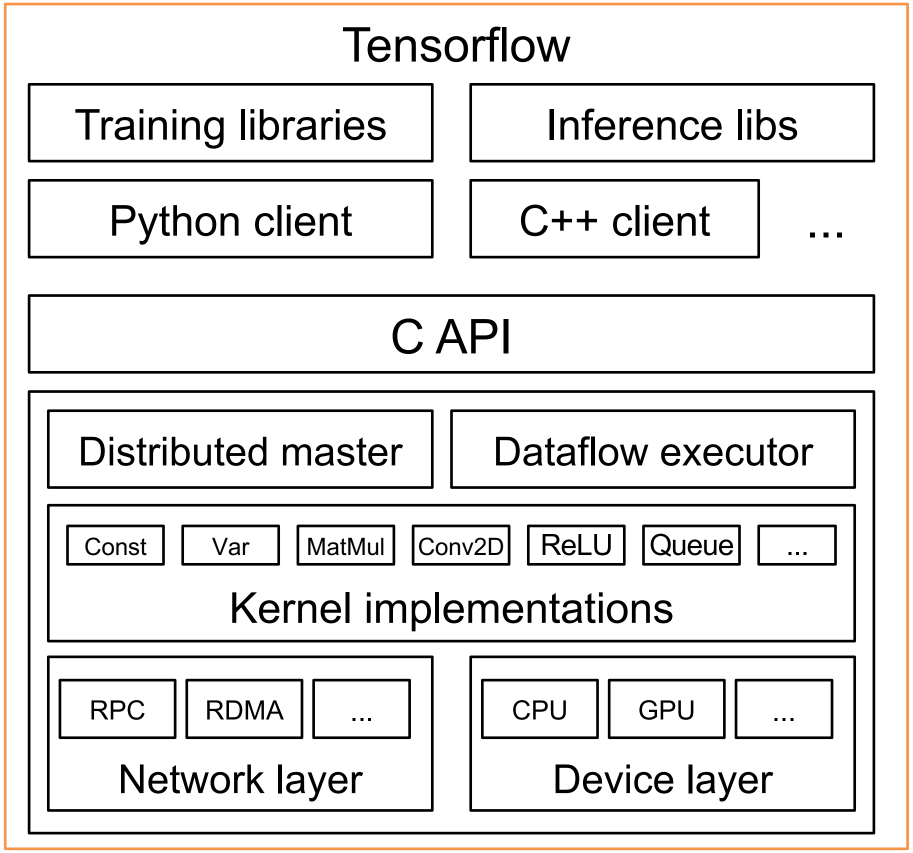


- Reduce error, amount of paramter
- Gain interpretability
- Gradient flow through different domains

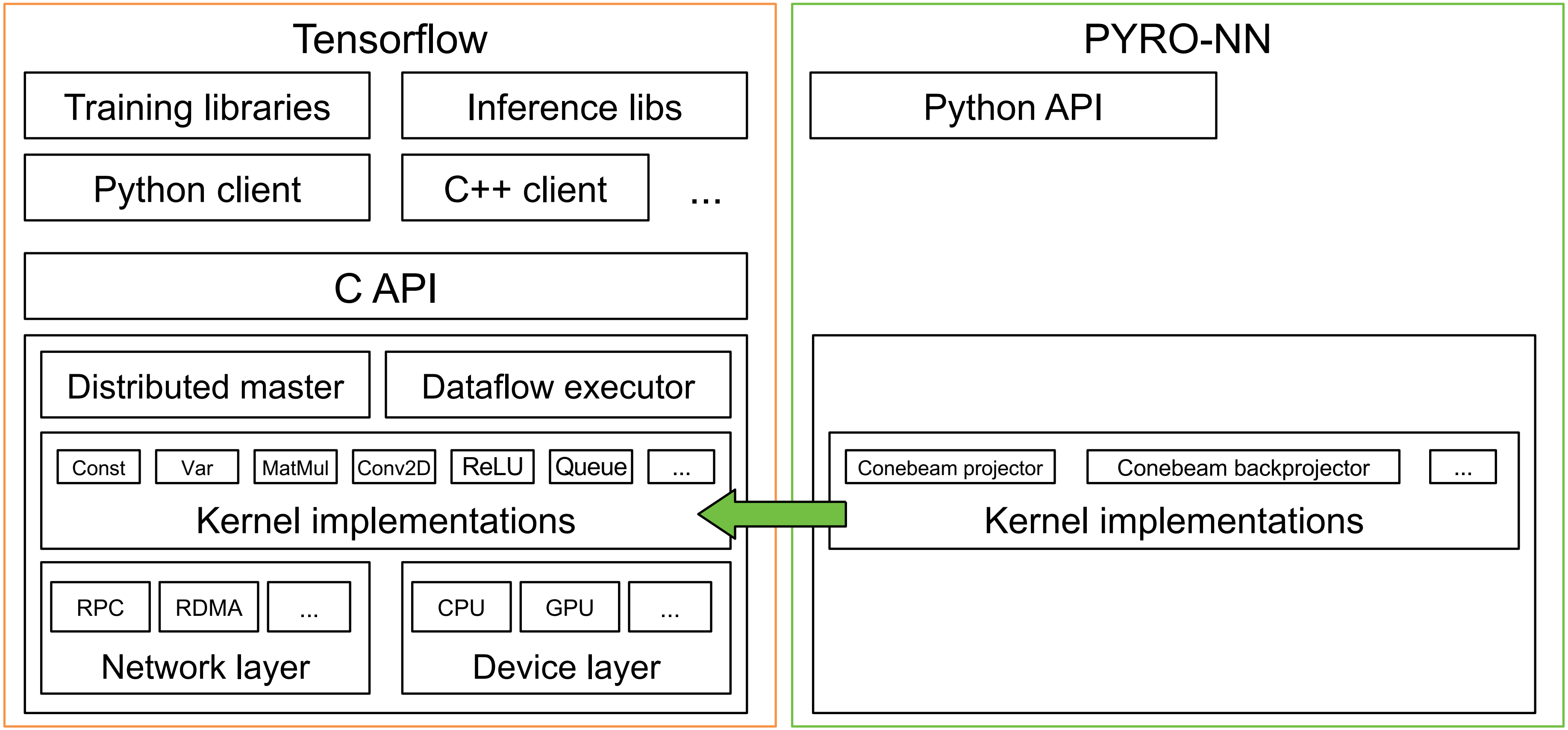
PYRO-NN: Python Reconstruction Operators in Neural Networks

- TF-Layers: Projector and back-projectors
  - 2D parallel, fan and 3D cone-beam
- Python API:
  - Layer abstraction
  - Geometries
  - Phantoms
  - Data generators

Operators in Tensorflow



Known Operators in Tensorflow


Tensorflow

Training libraries

Inference libs

Python client

C++ client

...

C API

Distributed master

Dataflow executor

Const | Var | MatMul | Conv2D | ReLU | Queue | ...

Kernel implementations

RPC | RDMA | ...

Network layer

CPU | GPU | ...

Device layer

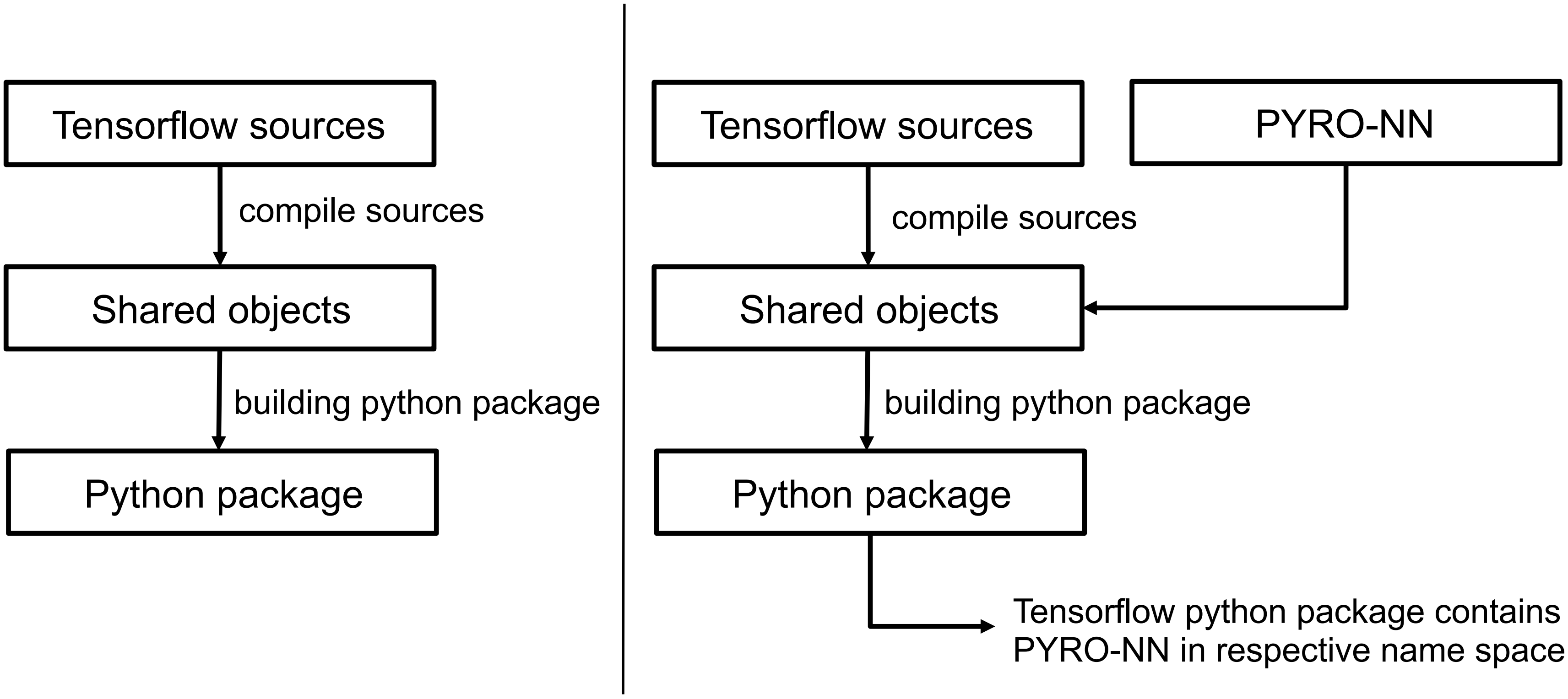
PYRO-NN

Python API

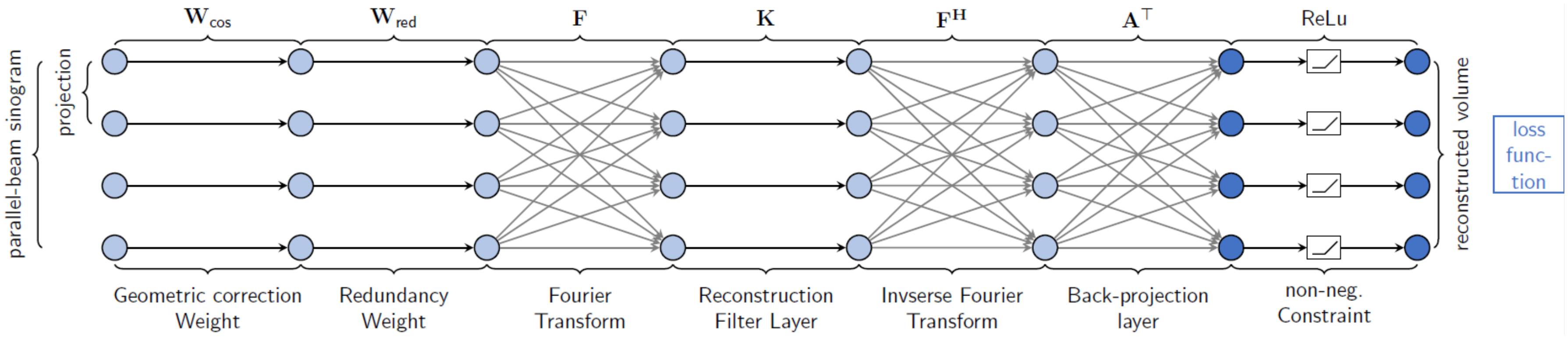
Conebeam projector | Conebeam backprojector | ...

Kernel implementations

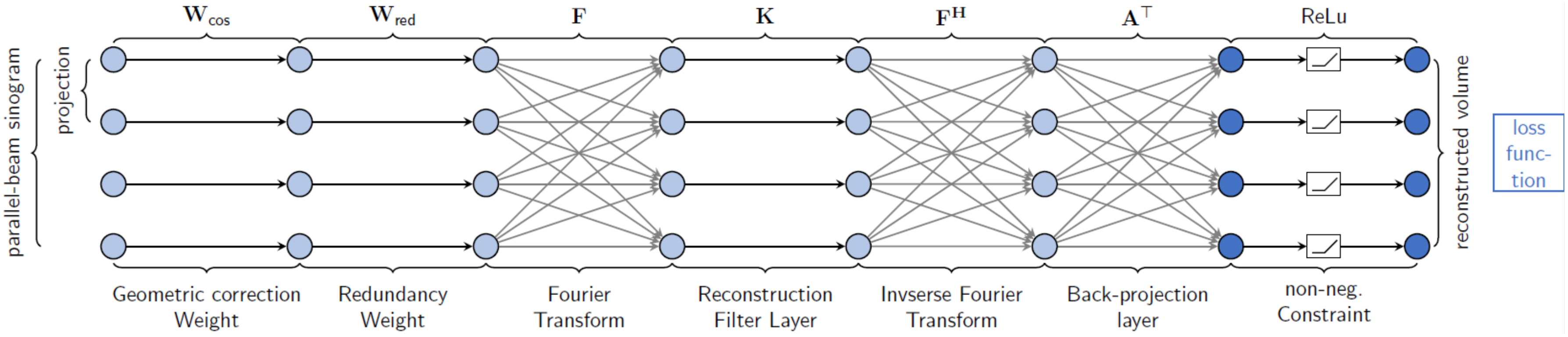
PYRO-NN Architecture



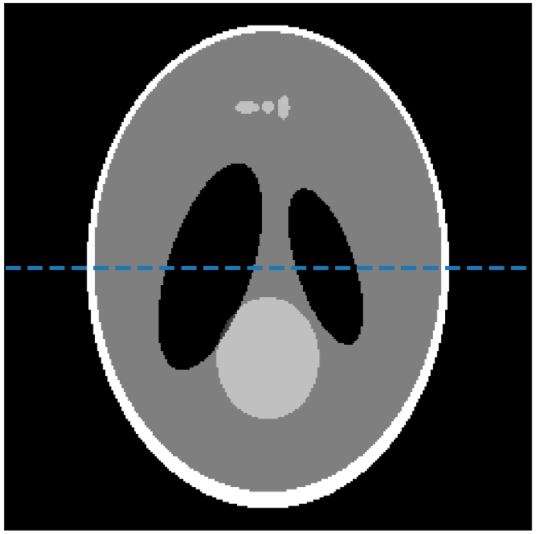
Short-Scan FDK-Net

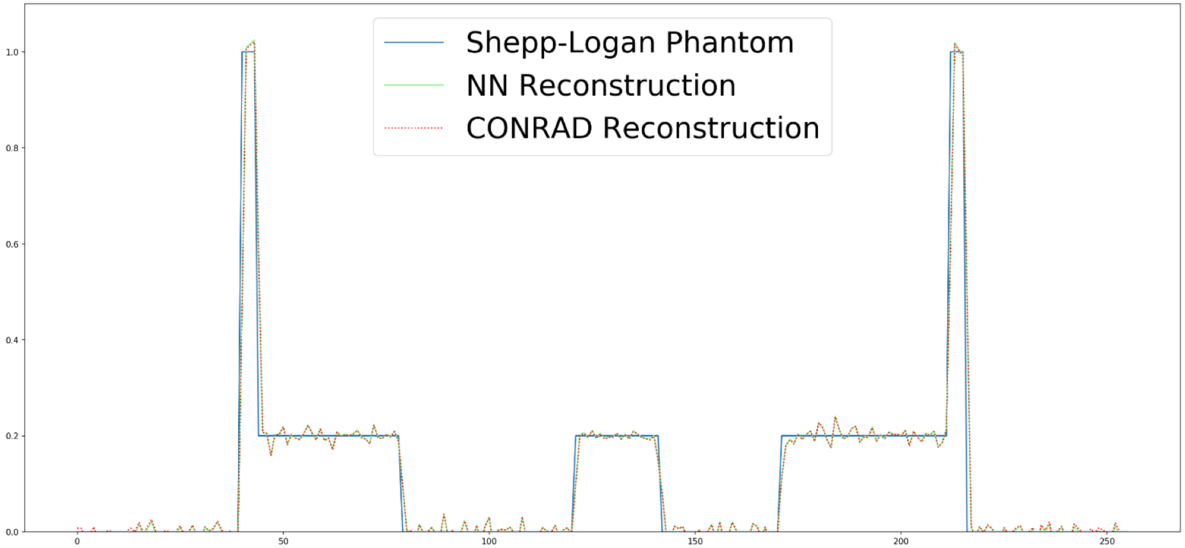


Short-Scan FDK-Net



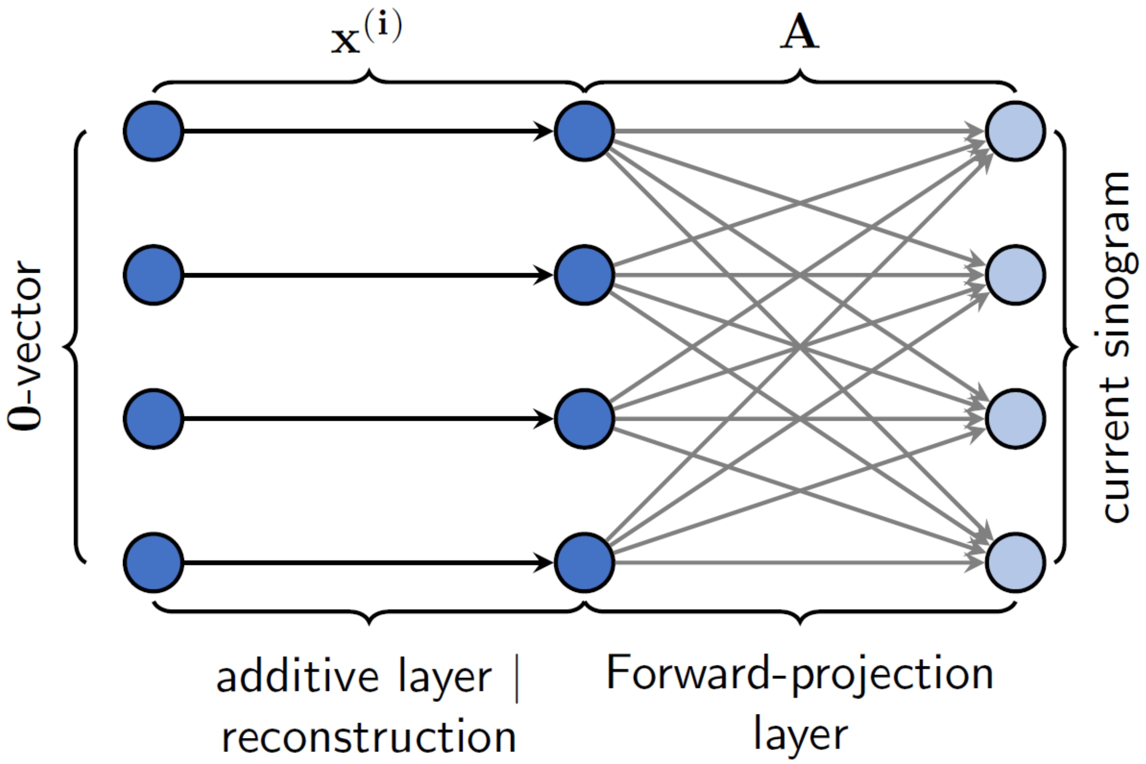
Central Slice NN Reconstruction





Iterative Reconstruction

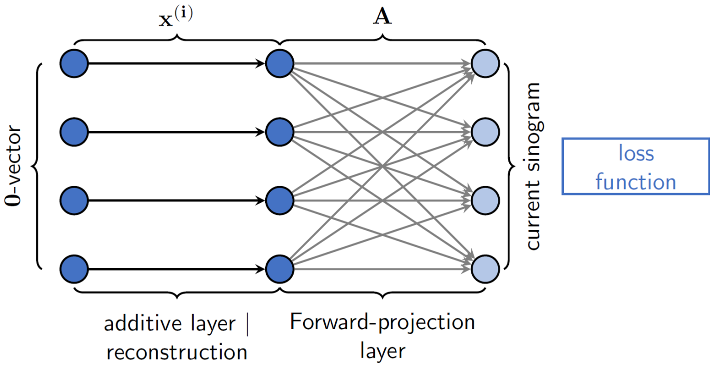
$$\min \ ||\mathbf{Ax} - \mathbf{p}||_2^2 + \lambda \mathbf{TV}(\mathbf{x})$$



loss function

Iterative Reconstruction

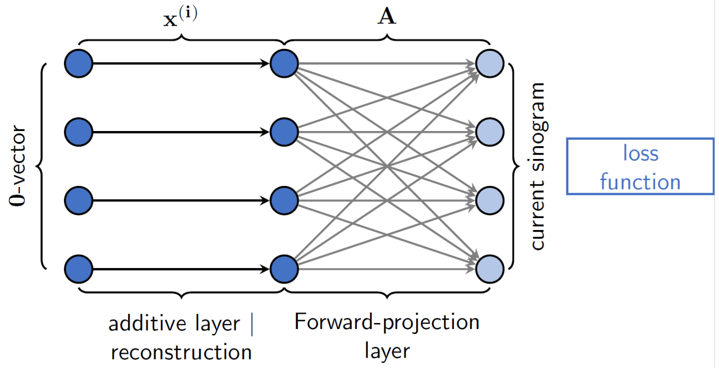
$$\min \;\; ||\mathbf{A}\mathbf{x} - \mathbf{p}||_2^2 + \lambda \mathbf{TV}(\mathbf{x})$$



```
from pyronn.ct_reconstruction.layers import projection_2d

reco = tf.get_variable(
                 initializer=np.zeroes(geometry.volume_shape),
                 trainable=True,
                 constraint=lambda x: tf.clip_by_value(x, 0, np.infty))
```

Iterative Reconstruction

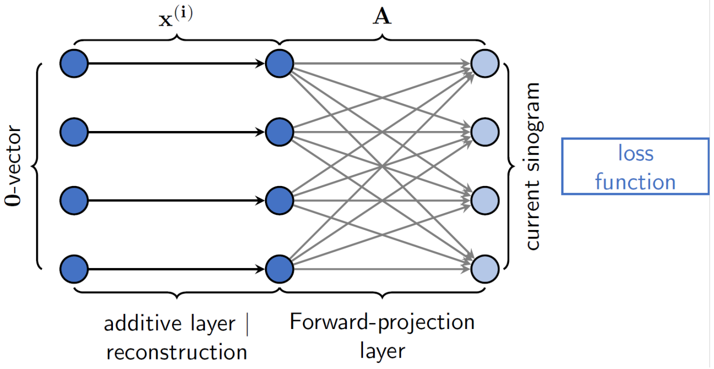
$$\min \; ||\mathbf{A}\mathbf{x} - \mathbf{p}||_2^2 + \lambda \mathbf{TV}(\mathbf{x})$$



```
from pyronn.ct_reconstruction.layers import projection_2d

reco = tf.get_variable(
                 initializer=np.zeroes(geometry.volume_shape),
                 trainable=True,
                 constraint=lambda x: tf.clip_by_value(x, 0, np.infty))

current_sino = projection_2d.parallel_projection2d(reco, geometry)
```

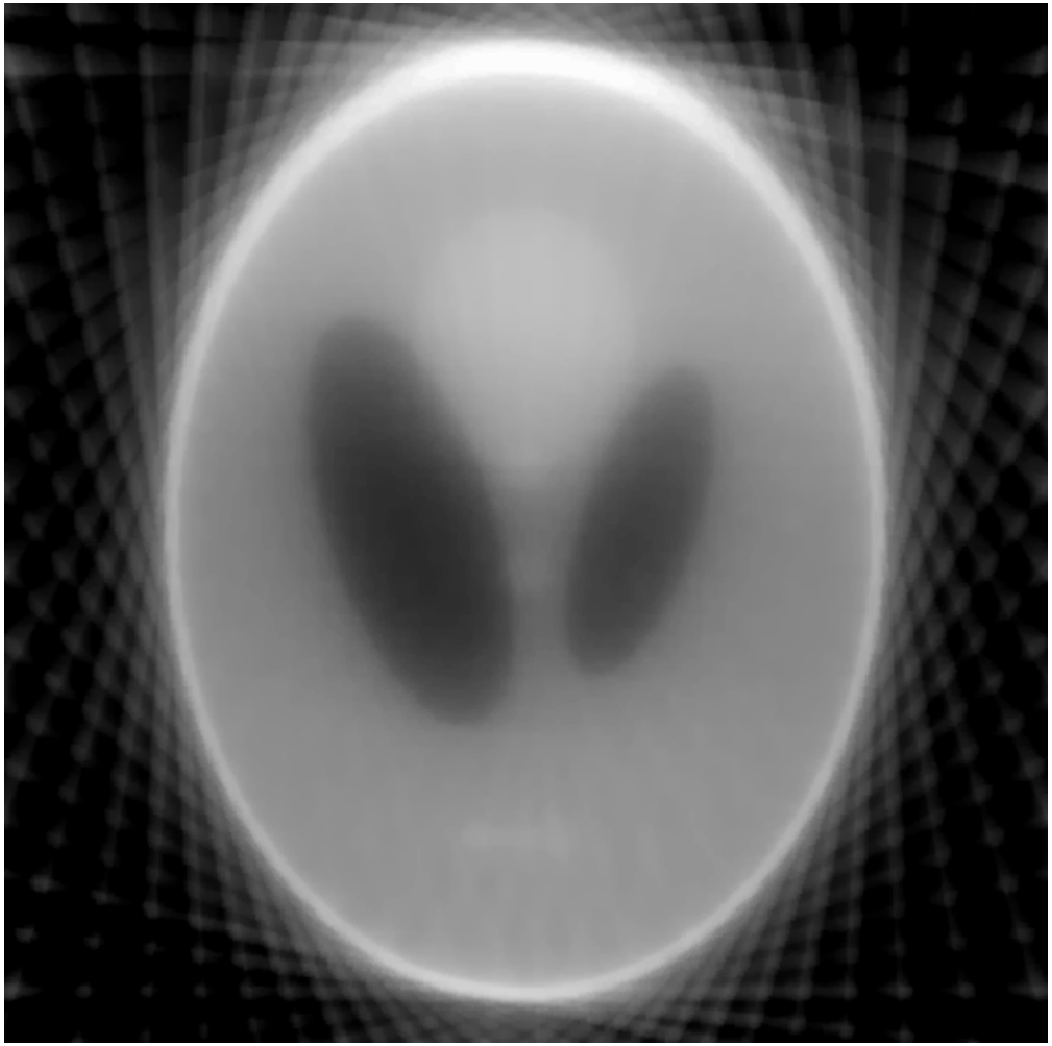
Iterative Reconstruction

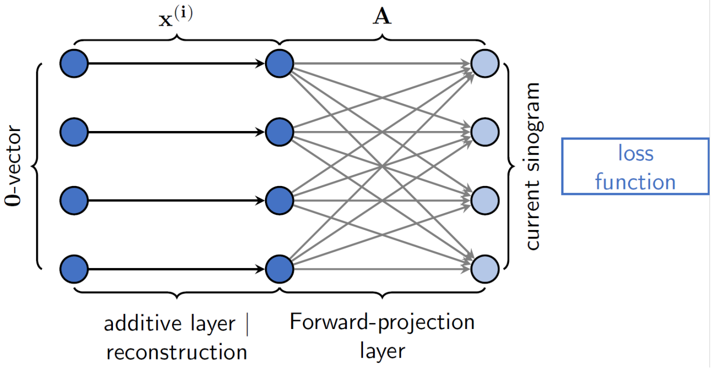
$$\min \; ||\mathbf{Ax} - \mathbf{p}||_2^2 + \lambda \mathbf{TV}(\mathbf{x})$$



```
from pyronn.ct_reconstruction.layers import projection_2d

reco = tf.get_variable(
                 initializer=np.zeroes(geometry.volume_shape),
                 trainable=True,
                 constraint=lambda x: tf.clip_by_value(x, 0, np.infty))

current_sino = projection_2d.parallel_projection2d(reco, geometry)

error = tf. squared_difference(current_sino,acquired_data)
           + lambda * tf.image.total_variation(reco)
```

Sparse View CT





PYRO-NN on GitHub

Python API:

https://github.com/csyben/PYRO-NN

Layers:

https://github.com/csyben/PYRO-NN-LAYERS

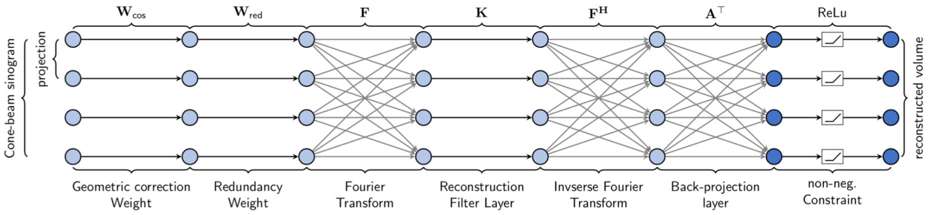
PYRO-NN

PYTHON RECONSTRUCTION OPERATORS FOR NEURAL NETWORKS

- Supports Tensorflow and PyTorch
- Full GPU Integration
- Open Source
- Apache 2.0 License

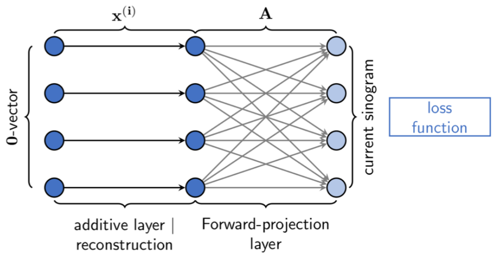
```
pip install pyronn
```

FBP Reconstruction



```
def model(self, sinogram):
    self.sinogram_cos = tf.multiply(sinogram, self.cosine_weight)
    self.redundancy_weighted_sino = tf.multiply(self.sinogram_cos,self.redundancy_weight)

    self.weighted_sino_fft = tf.fft(tf.cast(self.redundancy_weighted_sino, dtype=tf.complex64))
    self.filtered_sinogram_fft = tf.multiply(self.weighted_sino_fft, tf.cast(self.filter,dtype=tf.complex64))
    self.filtered_sinogram = tf.real(tf.ifft(self.filtered_sinogram_fft))

    self.reconstruction = cone_backprojection3d(self.filtered_sinogram,self.geometry, hardware_interp=True)

    return self.reconstruction, self.redundancy_weighted_sino
```

TV Recon



```
def model(self, input_volume):
    self.updated_reco = tf.add(input_volume, self.reco)
    self.current_sino = projection_2d.parallel_projection2d(self.updated_reco, self.geometry)
    return self.current_sino, self.reco
```

```
tv_loss_x = tf.image.total_variation(tf.transpose(self.current_reco))
tv_loss_y = tf.image.total_variation(self.current_reco)

self.loss = tf.reduce_sum(tf.squared_difference(self.label_element, self.current_sino)) + self.regularizer_weight*(tv_loss_x+tv_loss_y)
```