1 Introduction

The lme_cunstom_ops are layers for Tensorflow which can compute the forward projection and the back-projection of your object with parallel and fan-beam geometry. The forward projectors are implemented in CUDA on the graphics-card as ray-driven forward projector exploiting the texture interpolation capability of the gpu. The back-projectors are implemented as voxel-driven back-projectors using again the texture interpolation of the GPU.

2 API definition

2.1 Parallel Forward-projection Layer

```
lme_custom_ops.parallel_projection2d

volume              =   input_volume

volume_shape        =   [height, width]
                         <type:python list>

projection_shape    =   [number_of_projections, detector_width]
                        <type:python list>

volume_origin       =   [ volume_origin_y, volume_origin_x]
                         <type:tensor proto>

detector_origin     =   [ detector_origin_x ]
                        <type:tesnor proto>

volume_spacing      =   [ spacing_y, spacing_x]
                        <type: tensor proto

detector_spacing    =   [ detector_spacing_x]
                        <type: tensor proto>

ray_vectors         =   [ projection_vectors(y,x) ]
                        <type: tensor proto>
```

2.2 Parallel Back-projection Layer

```
lme_custom_ops.parallel_backprojection2d

volume              =   input_sinogram
```

```
sinogram_shape        = [number_of_projections, detector_width]
<type:python list>

volume_shape          =   [height, width]
<type:python list>

volume_origin         =   [ volume_origin_y, volume_origin_x]
<type:tensor proto>

detector_origin       =   [ detector_origin_x ]
<type:tesnor proto>

volume_spacing        =   [ spacing_y, spacing_x]
<type: tensor proto

detector_spacing      =   [ detector_spacing_x]
<type: tensor proto>

ray_vectors           =   [ projection_vectors(y,x) ]
<type: tensor proto>
```

2.3 Fan-beam Forward-projection Layer

```
lme_custom_ops.fan_projection2d

volume                =   input_volume

volume_shape          =   [height, width]
<type:python list>

projection_shape      =   [number_of_projections, detector_width]
<type:python list>

volume_origin         =   [ volume_origin_y, volume_origin_x]
<type:tensor proto>

detector_origin       =   [ detector_origin_x ]
<type:tesnor proto>

volume_spacing        =   [ spacing_y, spacing_x]
<type: tensor proto

detector_spacing      =   [ detector_spacing_x]
<type: tensor proto>

sid             =       source_iso_center_distance
```

```
<type: float>

sdd                 =       source_detector_distance
<type: float>

ray_vectors         =   [ projection_vectors(y,x) ]
<type: tensor proto>
```

2.4 Fan-beam Back-projection Layer

```
lme_custom_ops.fan_backprojection2d

volume              =   input

projection_shape    =   [number_of_projections, detector_width]
<type:python list>

volume_shape        =   [height, width]
<type:python list>

volume_origin       =   [ volume_origin_y, volume_origin_x]
<type:tensor proto>

detector_origin     =   [ detector_origin_x ]
<type:tesnor proto>

volume_spacing      =   [ spacing_y, spacing_x]
<type: tensor proto

detector_spacing    =   [ detector_spacing_x]
<type: tensor proto>

ray_vectors         =   [ projection_vectors(y,x) ]
<type: tensor proto>
```

2.5 Gradient Calculation

To use these layers in Networks the gradients need to be propagated through the network. Based on the back-propagation algorithm we need the derivative of the Layer. Since the derivative of the forward-projector ($\boldsymbol{A}$) is the back-projector ($\boldsymbol{A}^\top$). Thus, if we use a back-projection layer we need to tell Tensorflow that the gradient can be propagated using the forward-projection layer. This can be done by the following construct (example for the parallel back-projection):

```
from tensorflow.python.framework import ops

'''
Compute the gradient of the backprojection op
```

```
by invoking the forward projector.
'''
@ops.RegisterGradient( "ParallelBackprojection2D" )
def _backproject_grad( op, grad ):
proj = lme_custom_ops.parallel_projection2d(
volume                = grad,
volume_shape          = op.get_attr( "volume_shape" ),
projection_shape      = op.get_attr( "sinogram_shape" ),
volume_origin         = op.get_attr( "volume_origin" ),
detector_origin       = op.get_attr( "detector_origin" ),
volume_spacing        = op.get_attr( "volume_spacing" ),
detector_spacing      = op.get_attr( "detector_spacing" ),
ray_vectors           = op.get_attr( "ray_vectors" ),
)
return [ proj ]
```

3 Tensorflow related Hints and Examples

3.1 Tensorflow Session

```
with tf.Session() as sess:
        sess.run(...)
```

You need Tensorflow Session[1] to construct a graph of your network and to run the training process.

3.2 Placeholders

Due to the realization of Tensorflow you will need to use the placeholder construction if you want to change the value of a variable during the training procedure. There are different variables you want to change during the training e.g. your input and your label data or your learning rate (keyword: learning rate decay). Thus, in the following an example implementation for the learning rate:

```
import tensorflow as tf

1. learning_rate = tf.get_variable(name='learning_rate',
 dtype=tf.float32, initializer=tf.constant(0.0001), trainable=False)

2. learning_rate_placeholder =
 tf.placeholder(tf.float32, name='learning_rate_placeholder')

3. set_learning_rate =
 learning_rate.assign(self.learning_rate_placeholder)
```

[1] https://www.tensorflow.org/api_docs/python/tf/Session

1. with tf.get_variable we define a variable in tensorflow context. These variable needs a name and a type and can be initialized by an default value. Such a tensorflow variable can be set to trainable or non trainable. For the learning rate example we set the attribute trainable to False since we do not want to learn a learning rate.
2. As a second step we need to define a placeholder for this variable, the placeholder has its own name and the type has to be the same as the tensorflow variable.
3. The third step is to define a assign operation. With this operation we can feed a normal python variable to the Tensorflow context using the placeholder and assign the value to the variable we defined in step 1.

The variable can be then set with the following expression (you have to use your current active Tensorflow Session):

```
sess.run(set_learning_rate,
  feed_dict={learning_rate_placeholder: learning_rate})
```

So you run the assign operation defined in Step 3 and feed you python variable learning_rate into the placeholder created in Step 2 with the feed_dict.

4 Reconstruction/Implementation related Hints

To compute all necessary points and lines the easiest way is to setup a continuous coordinate system and discretize to the pixel at the last step. Therefore we need a way to switch between our pixel based and our continuous coordinate systems. For the reconstruction we assume or define that the volume we want to reconstruct or forward project is in the middle of our coordinate system.

Thus, we want to have our world coordinate system in the middle of our volume array. We have to consider that our arrays are normally zero based. Thus the middle of our array is the (width -1)/2.0 . Now we just have to incorporate the size of a pixel in millimetres to come to a coordinate system which describes our continuous real world:

```
origin = [ -(volume_height - 1) * spacing_y /2.0 ,
           -(volume_width - 1) * spacing_x / 2.0 ]
```

Some of the attributes of the provided layers need the tensor proto type. You can create such a type with the following command (shown with an example):

```
_detector_spacing = 0.5
detector_spacing =
 tf.contrib.util.make_tensor_proto([_detector_spacing], tf.float32)
```

4.1 PyConrad & SheppLogan

For visualization of your numpy array you can use pyconrad, which means you can handle you arrays with imagej.

You can start pyconrad with the following code:

```
import pyconrad as pyc

pyc.setup_pyconrad()
pyc.start_gui()
pyc.imshow(<your_numpy_array>,<A String for the window>)
```

you always need to call once setup_pyconrad() to initialize the communication with the java virtual machine. You can call imshow(...) to visualze any numpy array with imagej.
To get a 2D SheppLogan phantom you need the following lines of code:

```
import pyconrad as pyc

pyc.setup_pyconrad()

_ = pyconrad.ClassGetter('edu.stanford.rsl.tutorial.phantoms')

your_phantom =  _.SheppLogan(xy, False).as_numpy()
pyc.imshow(your_phantom, 'SheppLogan Phantom')
```