```
import numpy as np
import tensorflow as tf

LIMIT_A, LIMIT_B, EPSILON = -.1, 1.1, 1e-6

def weight(name, shape):
    return tf.get_variable(
            name=name,
            shape=shape,
            initializer=tf.contrib.layers.xavier_initializer()
    )

def bias(name, shape):
    return tf.get_variable(
            name=name,
            shape=shape,
            initializer=tf.constant_initializer(0.0)
    )

def conv(inputs, filter, stride, bias):
    if not bias:
        bias = 0
    return tf.nn.conv2d(
        input=inputs,
        filter=filter,
        strides=[1, stride, stride, 1],
        padding='SAME',
    ) + bias

def relu(inputs):
    return tf.nn.relu(inputs)

def pool(inputs, kernel_size, stride):
    return tf.nn.max_pool(
        value=inputs,
        ksize=[1, kernel_size, kernel_size, 1],
        strides=[1, stride, stride, 1],
        padding='SAME'
    )

def dense(inputs, filter, bias, activation=True):
    if not bias:
        bias = 0
    inputs = tf.layers.flatten(inputs)
    if activation:
        return tf.nn.relu(tf.matmul(inputs, filter) + bias)
    else:
        return tf.matmul(inputs, filter) + bias

class L0Conv2d():
    def __init__(self, scope, shape, droprate_init=.5, temperature=2./3., weight_decay=1.0, 
lambd=1.0, train=True):
        self.shape = shape
        self.temperature = temperature
        self.weight_decay = weight_decay
        self.lambd = lambd
        self.droprate_init = droprate_init
        self.dim_z = shape[3]

        with tf.variable_scope(scope):
            self.weights = tf.get_variable(
                    name='w',
                    shape=shape,
                    initializer=tf.initializers.he_normal()
            )
            self.bias = tf.get_variable(
                    name='b',
                    shape=self.dim_z,
                    initializer=tf.initializers.constant(0.0)
            )
            self.log_a = tf.get_variable(
                    name='log_a',
                    shape=self.dim_z,
                    initializer=tf.initializers.random_normal(np.log(1 - droprate_init) - 
np.log(droprate_init), 1e-2)
            )

    def constrain_parameters(self):
        return tf.clip_by_value(self.log_a, np.log(1e-2), np.log(1e2))

    def cdf_qz(self, x):
        xn = (x - LIMIT_A) / (LIMIT_B - LIMIT_A)
        logits = np.log(xn) - np.log(1 - xn)
        return tf.clip_by_value(tf.sigmoid(logits * self.temperature - self.log_a), EPSILON, 1 
- EPSILON)

    def quantile_concrete(self, x):
        y = tf.sigmoid((tf.log(x) - tf.log(1 - x) + self.log_a) / self.temperature)
        return y * (LIMIT_B - LIMIT_A) + LIMIT_A

    def regularization(self):
        q0 = self.cdf_qz(0)
        logpw_col = tf.reduce_sum(- (.5 * self.weight_decay * tf.pow(self.weights, 2)) - 
self.lambd, [2, 1, 0])
        logpw = tf.reduce_sum((1 - q0) * logpw_col)
        logpb = -tf.reduce_sum((1 - q0) * (.5 * self.weight_decay * tf.pow(self.bias, 2) - 
self.lambd))
        return logpw + logpb

    def count_l0(self):
        ppos = tf.reduce_sum(1 - self.cdf_qz(0))
        n = self.shape[0] * self.shape[1] * self.shape[2]
        return ppos

    def get_eps(self, shape):
        return tf.random_uniform(shape, EPSILON, 1-EPSILON)
        # return tf.Variable(tf.random_uniform(shape, EPSILON, 1-EPSILON))

    def hard_tanh(self, x, min_val, max_val):
        return tf.minimum(max_val, tf.maximum(min_val, x))

    def sample_z(self, batch_size, train=False):
        if train:
            eps = self.get_eps([batch_size, self.dim_z])
            z = tf.reshape(self.quantile_concrete(eps), [batch_size, 1, 1, self.dim_z])
            return self.hard_tanh(z, min_val=0.0, max_val=1.0)

        else:  # mode
            pi = tf.reshape(tf.sigmoid(self.log_a), [1, 1, 1, self.dim_z])
            return self.hard_tanh(pi * (LIMIT_B - LIMIT_A) + LIMIT_A, min_val=0.0, max_val=1.0)

    def sample_weights(self):
        z = tf.reshape(self.quantile_concrete(self.get_eps([self.dim_z])), [1, 1, 1, 
self.dim_z])
        return self.hard_tanh(z, min_val=0.0, max_val=1.0) * self.weights

class L0Dense():
    def __init__(self, scope, shape, droprate_init=.5, temperature=2./3., weight_decay=1.0, 
lambd=1.0, train=True):
        self.shape = shape
        self.temperature = temperature
        self.weight_decay = weight_decay
        self.lambd = lambd
        self.droprate_init = droprate_init

        with tf.variable_scope(scope):
            self.weights = tf.get_variable(
                    name='w',
                    shape=shape,
                    initializer=tf.initializers.he_normal()
            )
            self.bias = tf.get_variable(
                    name='b',
                    shape=self.shape[1],
                    initializer=tf.initializers.constant(0.0)
            )
            self.log_a = tf.get_variable(
                    name='log_a',
                    shape=self.shape[0],
                    initializer=tf.initializers.random_normal(np.log(1 - droprate_init) - 
np.log(droprate_init), 1e-2)
            )

    def constrain_parameters(self):
        return tf.clip_by_value(self.log_a, np.log(1e-2), np.log(1e2))

    def cdf_qz(self, x):
        xn = (x - LIMIT_A) / (LIMIT_B - LIMIT_A)
        logits = np.log(xn) - np.log(1 - xn)
        return tf.clip_by_value(tf.sigmoid(logits * self.temperature - self.log_a), EPSILON, 1 
- EPSILON)

    def quantile_concrete(self, x):
        y = tf.sigmoid((tf.log(x) - tf.log(1 - x) + self.log_a) / self.temperature)
        return y * (LIMIT_B - LIMIT_A) + LIMIT_A

    def regularization(self):
        q0 = self.cdf_qz(0)
        logpw_col = tf.reduce_sum(- (.5 * self.weight_decay * tf.pow(self.weights, 2)) - 
self.lambd, 1)
        logpw = tf.reduce_sum((1 - q0) * logpw_col)
        logpb = -tf.reduce_sum(.5 * self.weight_decay * tf.pow(self.bias, 2))
        return logpw + logpb

    def count_l0(self):
        ppos = tf.reduce_sum(1 - self.cdf_qz(0))
        n = self.shape[1]
        return ppos

    def get_eps(self, shape):
        return tf.random_uniform(shape, EPSILON, 1-EPSILON)
        # return tf.Variable(tf.random_uniform(shape, EPSILON, 1-EPSILON))

    def hard_tanh(self, x, min_val, max_val):
        return tf.minimum(max_val, tf.maximum(min_val, x))

    def sample_z(self, batch_size, train=False):
        if train:
            eps = self.get_eps([batch_size, self.shape[0]])
            z = self.quantile_concrete(eps)
            return self.hard_tanh(z, min_val=0.0, max_val=1.0)

        else:  # mode
            pi = tf.tile(tf.reshape(tf.sigmoid(self.log_a), [1, self.shape[0]]), 
tf.constant([batch_size, 1]))
            return self.hard_tanh(pi * (LIMIT_B - LIMIT_A) + LIMIT_A, min_val=0.0, max_val=1.0)

    def sample_weights(self):
        z = self.quantile_concrete(self.get_eps([self.shape[0]]))
        return tf.reshape(self.hard_tanh(z, min_val=0.0, max_val=1.0), [self.shape[0], 1]) * 
self.weights

    def _sample_weights(self):
        z = self.quantile_concrete(self.get_eps([self.shape[1]]))
        return tf.reshape(self.hard_tanh(z, min_val=0.0, max_val=1.0), [1, self.shape[1]]) *
self.weights
```