기본과제 CNN Tensorflow 구현

In [1]:

```
%env CUDA_VISIBLE_DEVICES=1

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import time
import argparse
import numpy as np
import tensorflow as tf

print('Tensorflow version ' + tf.__version__)
```

```
env: CUDA_VISIBLE_DEVICES=1
Tensorflow version 1.3.0
```

전체 레이어에서 공통으로 사용되는 Tensorboard summary 함수

- 각 레이어 class에서 상속하기 위해 static method로 정의함

In [2]:

```
class Summary(object):
  @staticmethod
  def _summary(name, var):
    _mean = tf.reduce_mean(var)
    _variance = tf.reduce_mean(tf.square(var - _mean))
    tf.summary.scalar(name+'_mean', _mean)
    tf.summary.scalar(name+'_variance', _variance)
    tf.summary.histogram(name, var)
```

Convolution layer class

- 2D convolution layer 구현
- 입력된 BHWD 이미지 데이터에 대해 요청된 actication 결과를 돌려줌
- 요청된 layer 크기에 맞춰 kernel weight와 bias를 생성하되, batch normalization이 적용될 경우는 bias를 생성하지 않음
- 각 weight와 bias 생성시 Tensorboard summary에 각 변수에 대한 평균, 분산, 히스토그램이 추가됨

In [3]:

```
class ConvLayer(Summary):
  '''
  Construct a convolutional 2D layer and its summaries
  args:
    image  = input image array
    ch_in  = input image channel size (e.g. rgb = 3)
    ch_out = output channel size (number of kernels)
    size   = size of kernel (patch)
    stride = kernel (patch) stride
    activation = activation function (cf. 'bn': for batch normalization)
  '''
  def __init__(self, image, ch_in, ch_out, size, stride, activation='none'):
    self.img = image
    self.strd = stride
    self.act = activation.lower()
    _W_shape = [size, size, ch_in, ch_out]
    self.W = tf.Variable(tf.truncated_normal(_W_shape, stddev=0.1), trainable=True, name='W')
    self._summary('W', self.W)
    if self.act != 'bn':
      self.B = tf.Variable(tf.constant(0.1, tf.float32, [ch_out]), trainable=True, name='B')
      self._summary('B', self.B)

  def out(self):
    WX = tf.nn.conv2d(self.img, self.W, strides=[1, self.strd, self.strd, 1], padding='SAME')
    if self.act == 'relu':
```

```
    if self.act == 'relu':
      return tf.nn.relu(WX + self.B)
    elif self.act == 'bn':
      return WX
    elif self.act == 'none':
      return WX + self.B
    else:
      raise ValueError('ERROR: unsupported activation option')
```

Fully connected layer class

- Fully connected layer 구현
- 기능 및 구조는 ConvLayer와 유사함

In [4]:

```
class FCLayer(Summary):
  '''
  Construct a fully connected layer and its summaries
  args:
    input_ = input array
    n_in   = input size
    n_out  = output size
    activation = activation function
  '''
  def __init__(self, input_, n_in, n_out, activation='none'):
    self.input_ = input_
    self.n_in = n_in
    self.n_out = n_out
    self.act = activation.lower()
    self.W = tf.Variable(tf.truncated_normal([n_in, n_out], stddev=0.1), trainable=True, name='W')
    self._summary('W', self.W)
    self.B = tf.Variable(tf.constant(0.0, tf.float32, [n_out]), trainable=True, name='B')
    self._summary('B', self.B)

  def out(self):
    if self.act == 'relu':
      return tf.nn.relu(tf.matmul(self.input_, self.W) + self.B)
    elif self.act == 'none':
      return tf.matmul(self.input_, self.W) + self.B
    else:
      raise ValueError('ERROR: unsupported activation option')
```

Batch normalization class

- 입력에 대한 batch normalization을 결과를 돌려줌
- training 시에는 입력된 각 batch의 평균과 분산을 normalization에 사용하고, test 시에는 기존 training의 전체 입력데이터에 대한 평균과 분산의 추정치(Exponential Moving Average)를 사용함

In [5]:

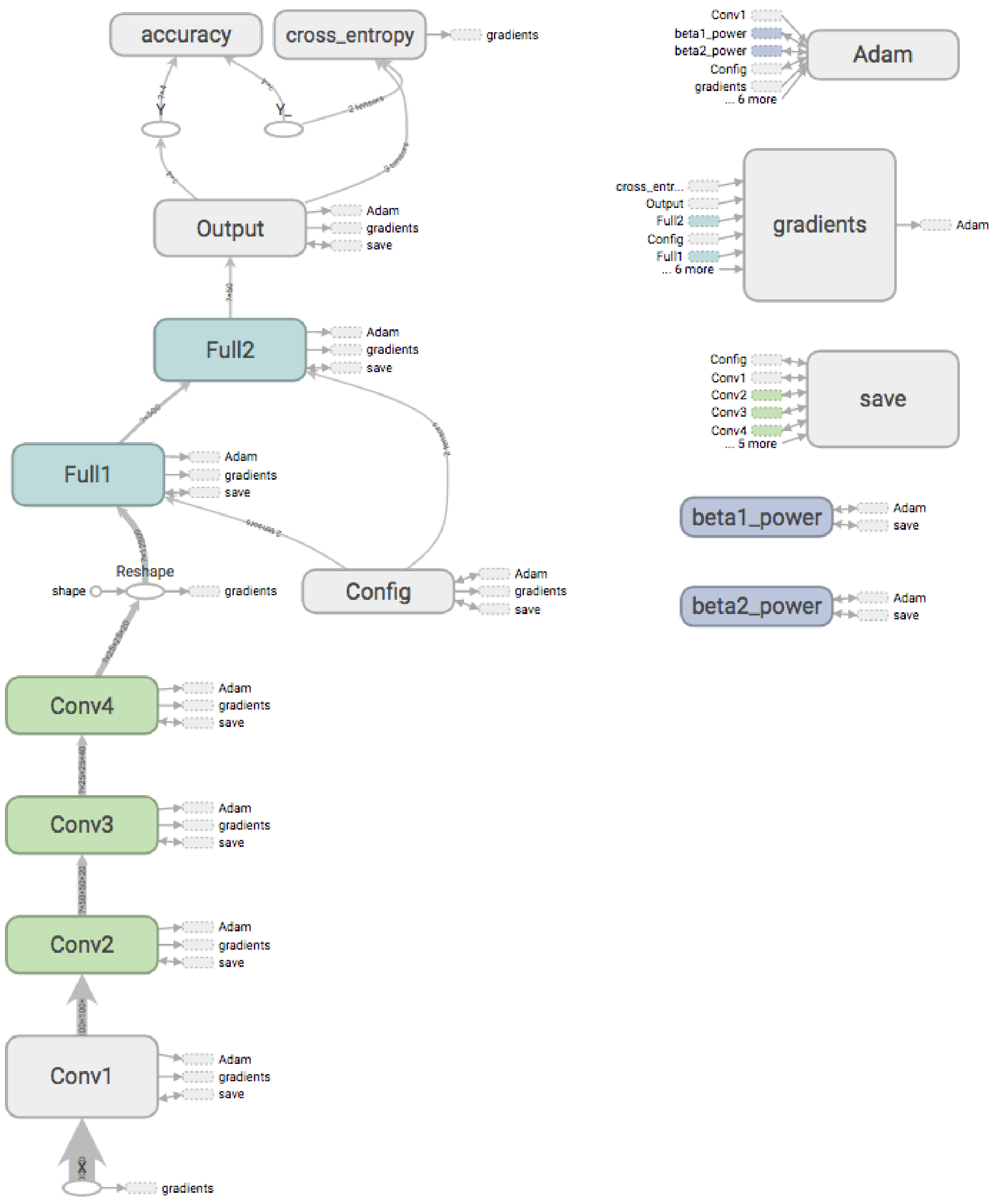
```
class BatchNorm(Summary):
  '''
  Construct a batch normalization for input array
  args
    input_ = input array tensor
    n_out  = output size
    train  = True: train phase, False: test phase
    activation = activation function
  '''
  def __init__(self, input_, n_out, train, activation='none'):
    self.input_ = input_
    self.act = activation.lower()
    self.beta = tf.Variable(tf.constant(0.0, shape=[n_out]), trainable=True, name='beta')
    self._summary('beta', self.beta)
    self.gamma = tf.Variable(tf.constant(1.0, shape=[n_out]), trainable=True, name='gamma')
    self._summary('gamma', self.gamma)
    batch_mean, batch_var = tf.nn.moments(input_, [0,1,2], name='moments')
    ema = tf.train.ExponentialMovingAverage(decay=0.5)
    ema_apply_op = ema.apply([batch_mean, batch_var])
    if train:
      with tf.control_dependencies([ema_apply_op]):
        self.mean, self.var = tf.identity(batch_mean), tf.identity(batch_var)
    else:
```

```
    self.mean, self.var = ema.average(batch_mean), ema.average(batch_var)

  def out(self):
    norm = tf.nn.batch_normalization(self.input_, self.mean, self.var, self.beta, self.gamma, 1e-3)
    if self.act == 'relu':
      return tf.nn.relu(norm)
    elif self.act == 'none':
      return norm
    else:
      raise ValueError('ERROR: unsupported activation option')
```

CNN model 정의

- Input image는 1000 x 1000 해상도 rgb color png파일
- Convolution layer 4개, fully connected layer 2개로 구성
- 첫번째 convolution layer는 batch normalization 적용
- 각 fully connected layer는 drop out 적용 (keep probability 75%)
- 모든 layer의 activation function은 ReLU 사용



In [6]:

```
def model(X, Y_, p_keep=None):
  '''
```

```
Define a DNN inference model
args:
  X      = input image array
  Y_     = labels of input (solutions of Y)
  train  = True: train phase, False: test phase
  p_keep = keep probability of drop out, if NOT defined TEST phase model will run
returns:
  Y      = predicted output array (e.g. [1, 0, 0, 0])
  cross_entropy
  accuracy
  incorrects = indices of incorrect inference

X     : [1000 x 1000 x 3] HWC image volume
Conv1 : [100 x 100 x K1] output volume after [100 x 100 x 3] kernel with stride 10
Conv2 : [50 x 50 x K2] output volume after [10 x 10 x K1] kernel with stride 2
Conv3 : [25 x 25 x K3] output volume after [5 x 5 x K2] kernel with stride 2
Conv4 : [25 x 25 x K4] output volume after [3 x 3 x K3] kernel with stride 1
Full1 : [F1] output nodes from [25 * 25 * K4] input nodes
Full2 : [F2] output nodes from [F1] input nodes
Output: [4] ouput nodes from [F2] input nodes
'''
K1 = 10    # Conv1 layer feature map depth
K2 = 20    # Conv2 layer feature map depth
K3 = 40    # Conv3 layer feature map depth
K4 = 20    # Conv4 layer feature map depth
F1 = 500   # Full1 layer node size
F2 = 50    # Full2 layer node size

train_phase = False if p_keep is None else True

with tf.variable_scope('Conv1'):
  y1 = ConvLayer(X, 3, K1, 100, 10, activation='BN').out()
  with tf.variable_scope('BN'):
    y1 = BatchNorm(y1, K1, train_phase, activation='ReLU').out()
with tf.variable_scope('Conv2'):
  y2 = ConvLayer(y1, K1, K2, 10, 2, activation='ReLU').out()
with tf.variable_scope('Conv3'):
  y3 = ConvLayer(y2, K2, K3, 5, 2, activation='ReLU').out()
with tf.variable_scope('Conv4'):
  y4 = ConvLayer(y3, K3, K4, 3, 1, activation='ReLU').out()
y4_rs = tf.reshape(y4, shape=[-1, 25*25*K4])

with tf.variable_scope('Full1'):
  y5 = FCLayer(y4_rs, 25*25*K4, F1, activation='ReLU').out()
  if train_phase: y5 = tf.nn.dropout(y5, p_keep)
with tf.variable_scope('Full2'):
  y6 = FCLayer(y5, F1, F2, activation='ReLU').out()
  if train_phase: y6 = tf.nn.dropout(y6, p_keep)
with tf.variable_scope('Output'):
  Ylogits = FCLayer(y6, F2, 4).out()
Y = tf.nn.softmax(Ylogits, name='Y')

with tf.variable_scope('cross_entropy'):
  cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=Ylogits, labels=Y_)
  cross_entropy = tf.reduce_mean(cross_entropy)
with tf.variable_scope('accuracy'):
  correct_prediction = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  incorrects = tf.squeeze(tf.where(tf.logical_not(correct_prediction)), [1])

return Y, cross_entropy, accuracy, incorrects
```

Tensorflow runtime sessions

1. png image를 array data로 변환

- input image는 png 파일 포멧으로 [모터속도Hz]-[토크%]-[수집일련번호]-[상태번호] 의 이름을 가짐 (예: 25Hz-50%-251-0.png)
- [상태번호]는 0: 정상, 1: stator fault, 2: rotor fault, 3: bearing fault로 정의
- 지정된 directory에서 전체 png 파일 목록을 가져온 후, 각 파일의 path를 ***png*** array에 저장하고, 해당 파일의 이미지 raw 데이터를 array로 변환하여 ***X*** array에 저장, 파일명으로 부터 상태번호를 추출하여 ***Y_*** array에 저장

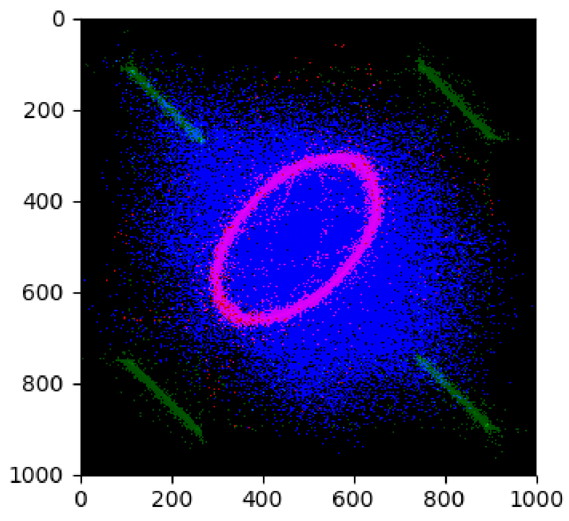
In [7]:

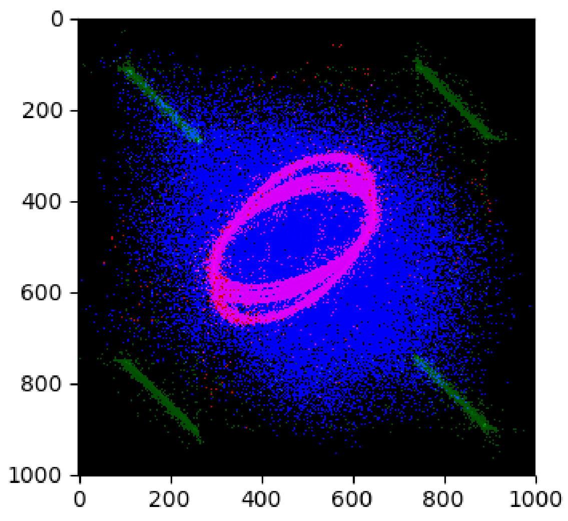
```
def gen_data(file_path, batch_size=1, one_hot=True, shuffle=True):
```

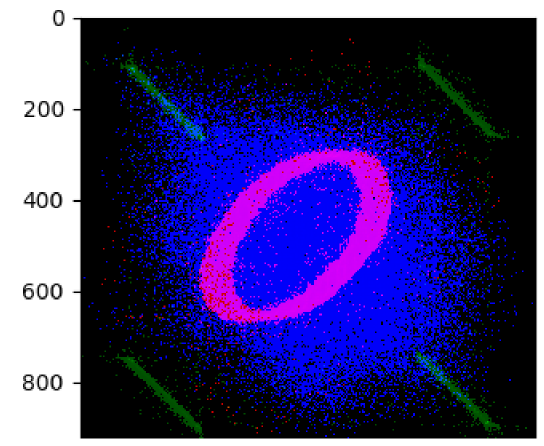
```
def gen_data(file_path, batch_size=1, one_hot=True, shuffle=True):
  '''
  Generate input data batches from png images
  args
    file_path  = input image(png) file path
    batch_size = batch size for each training step
    one_hot    = True: output labels (Y_) as one_hot arrays
    shuffle    = True: shuffle data queue sequence
  return
    data = {'X':[X1, ..., Xn], 'Y_':[Y_1, ..., Y_n], 'png':[png_path1, ..., png_pathn]}
  '''
  data = {}
  files = [os.path.join(file_path, s) for s in os.listdir(file_path)]
  queue = tf.train.string_input_producer(files, shuffle=shuffle)
  reader = tf.WholeFileReader()
  file_path, contents = reader.read(queue)
  img = tf.image.decode_png(contents, channels=3)
  with tf.Session() as sess:
    tf.global_variables_initializer().run()
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)
    raw = [sess.run([file_path, img]) for _ in files]
    data['png'] = [f_i[0].decode() for f_i in raw]
    data['X'] = [f_i[1] for f_i in raw]
    data['Y_'] = [int(p.strip('.png').split('-')[3]) for p in data['png']]
    if one_hot: data['Y_'] = sess.run(tf.one_hot(data['Y_'],4))
    coord.request_stop()
    coord.join(threads)
  return data
```

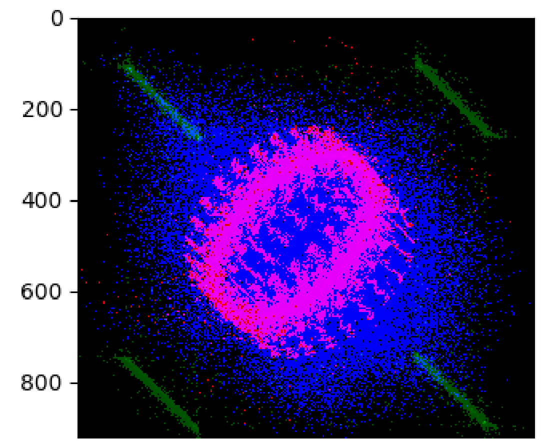
Input image 예시

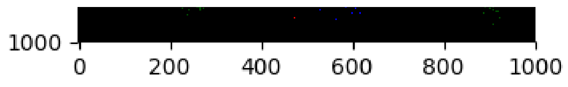
- 상단 좌측부터 우측으로 정상, stator fault, rotor fault, bearing fault 순
  - Red: 전류 2채널 (phase A, B) raw data에 대한 correlation matrix image
  - Green: 전압 2채널 (phase A, B) raw data에 대한 correlation matrix image
  - Blue: 진동 2채널 (axis y, z) raw data에 대한 correlation matrix image
- [참고] 각 이미지는 사람이 확인하기 쉽도록 실제 이미지보다 밝기가 증가되어 있음











2. training phase

- adam optimizer 사용
- learning rate는 0.001 에서 시작하여 5단계에 걸친 exponential decay 적용

In [8]:

```
def do_train(MAX_STEP, BATCH_SIZE, INPUT_PATH, MODEL_PATH, LOG_DIR):
    startTime = time.time()
    # input generation
    with tf.Graph().as_default() as input_g:
      data = gen_data(INPUT_PATH, BATCH_SIZE)
      n_data = len(data['png'])
      print('[%6.2f] successfully generated train data: %d samples'%(time.time()-startTime, n_data))

    # training phase
    with tf.Graph().as_default() as train_g:
      # input X: 1000 x 1000 rgb color image
      X = tf.placeholder(tf.float32, [None, 1000, 1000, 3], name='X')
      # target values Y_: 0=normal, 1=stator fault, 2=rotor fault, 3=bearing fault
      Y_ = tf.placeholder(tf.float32, [None, 4], name='Y_')
      with tf.variable_scope('Config'):
        # dropout keep probability
        p_keep = tf.placeholder(tf.float32, name='p_keep')
        # learning rate
        global_step = tf.Variable(0, name='global_step', trainable=False)
        lr = tf.train.exponential_decay(0.001, global_step, int(MAX_STEP/5), 0.5, staircase=True, name='lr')

      # load inference model
      Y, cross_entropy, accuracy, _ = model(X, Y_, p_keep)
      train_op = tf.train.AdamOptimizer(lr).minimize(cross_entropy, global_step=global_step)

      with tf.variable_scope('Metrics'):
        tf.summary.scalar('accuracy', accuracy)
        tf.summary.scalar('cross_entropy', cross_entropy)
        tf.summary.scalar('learning_rate', lr)

      # set tensorboard summary and saver
      merged = tf.summary.merge_all()
      saver = tf.train.Saver(max_to_keep=100)

      # training session
      print('----- training start -----')
      with tf.Session() as sess:
        sum_writer = tf.summary.FileWriter(LOG_DIR, sess.graph)
        tf.global_variables_initializer().run()
        step = 1
        while step <= MAX_STEP:
          for batch in range(int(n_data/BATCH_SIZE)+1):
            s = batch*BATCH_SIZE
            e = (batch+1)*BATCH_SIZE if (batch+1)*BATCH_SIZE < n_data else n_data
            if e <= s: break
            _, summary, acc, ent = sess.run([train_op, merged, accuracy, cross_entropy],
                                            {X:data['X'][s:e], Y_:data['Y_'][s:e], p_keep:0.75})
            sum_writer.add_summary(summary, step)
            print('[%6.2f] step:%3d, size:%3d, lr:%f, accuracy:%f, cross entropy:%f'
                  %(time.time()-startTime, step, e-s, lr.eval(), acc, ent))
            if (MAX_STEP-step)<10 or step%100==0:
              saver.save(sess, MODEL_PATH, global_step=step)
            step += 1
            if step > MAX_STEP: break
      print('-----  training end  -----')
```

실행 예

- *INPUT*: training에 사용할 png image directory 경로 지정
- *SAVE*: training이 완료된 network parameter 저장경로 지정
- *LOG*: Tensorboard log 저장경로 지정

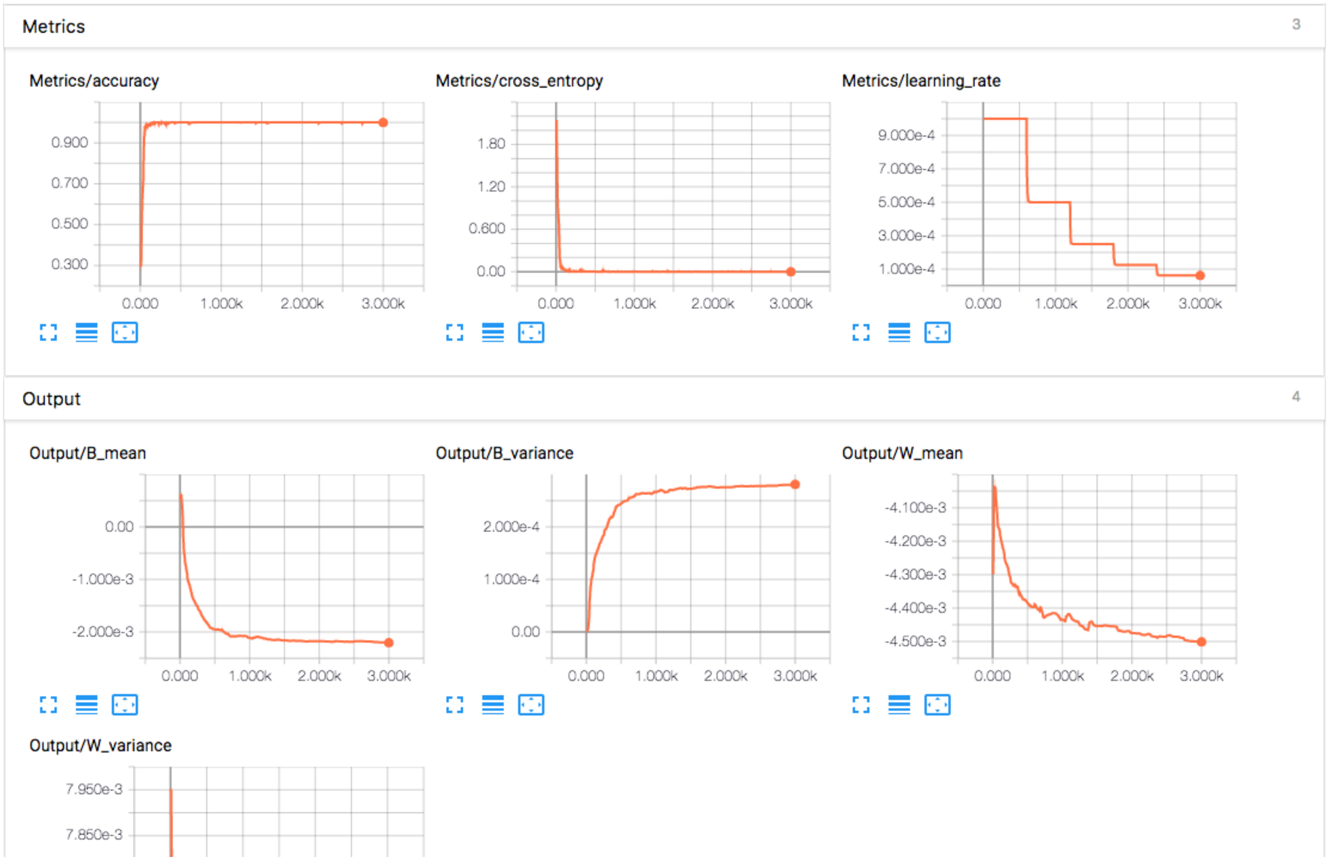
In [10]:

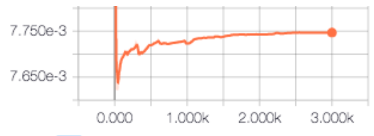
```
INPUT='/mnt/data/camus/images/20170919/'
SAVE='/mnt/data/camus/project/tmp/train/model/ckpt'
LOG='/mnt/data/camus/project/tmp/train/log/'

# do train with batch size 120 and maximum step 30
do_train(30, 120, INPUT, SAVE, LOG)
```

```
[ 13.23] successfully generated train data: 1184 samples
----- training start -----
[282.15] step:  1, size:120, lr:0.001000, accuracy:0.216667, cross entropy:27.592012
[284.63] step:  2, size:120, lr:0.001000, accuracy:0.300000, cross entropy:6.706369
[287.11] step:  3, size:120, lr:0.001000, accuracy:0.241667, cross entropy:6.977856
[289.58] step:  4, size:120, lr:0.001000, accuracy:0.300000, cross entropy:3.377185
[292.06] step:  5, size:120, lr:0.001000, accuracy:0.300000, cross entropy:2.045619
[294.53] step:  6, size:120, lr:0.000500, accuracy:0.325000, cross entropy:1.575463
[297.01] step:  7, size:120, lr:0.000500, accuracy:0.275000, cross entropy:1.592177
[299.50] step:  8, size:120, lr:0.000500, accuracy:0.275000, cross entropy:1.564621
[301.97] step:  9, size:120, lr:0.000500, accuracy:0.291667, cross entropy:1.513112
[565.96] step: 10, size:104, lr:0.000500, accuracy:0.298077, cross entropy:1.492568
[568.48] step: 11, size:120, lr:0.000500, accuracy:0.375000, cross entropy:1.382447
[570.97] step: 12, size:120, lr:0.000250, accuracy:0.333333, cross entropy:1.345186
[573.46] step: 13, size:120, lr:0.000250, accuracy:0.383333, cross entropy:1.369876
[575.95] step: 14, size:120, lr:0.000250, accuracy:0.383333, cross entropy:1.362009
[578.84] step: 15, size:120, lr:0.000250, accuracy:0.375000, cross entropy:1.393385
[581.38] step: 16, size:120, lr:0.000250, accuracy:0.341667, cross entropy:1.378198
[583.85] step: 17, size:120, lr:0.000250, accuracy:0.341667, cross entropy:1.367950
[586.32] step: 18, size:120, lr:0.000125, accuracy:0.341667, cross entropy:1.335805
[588.80] step: 19, size:120, lr:0.000125, accuracy:0.358333, cross entropy:1.334626
[590.96] step: 20, size:104, lr:0.000125, accuracy:0.278846, cross entropy:1.394738
[593.45] step: 21, size:120, lr:0.000125, accuracy:0.350000, cross entropy:1.323875
[596.02] step: 22, size:120, lr:0.000125, accuracy:0.366667, cross entropy:1.289862
[598.57] step: 23, size:120, lr:0.000125, accuracy:0.416667, cross entropy:1.304380
[601.13] step: 24, size:120, lr:0.000063, accuracy:0.341667, cross entropy:1.328330
[603.69] step: 25, size:120, lr:0.000063, accuracy:0.400000, cross entropy:1.319643
[606.24] step: 26, size:120, lr:0.000063, accuracy:0.433333, cross entropy:1.305296
[608.80] step: 27, size:120, lr:0.000063, accuracy:0.316667, cross entropy:1.351926
[611.36] step: 28, size:120, lr:0.000063, accuracy:0.275000, cross entropy:1.369204
[613.93] step: 29, size:120, lr:0.000063, accuracy:0.383333, cross entropy:1.344195
[616.16] step: 30, size:104, lr:0.000031, accuracy:0.326923, cross entropy:1.344097
-----  training end  -----
```

- 전체 학습 횟수(max step) **3000** 정도에서 전체 parameter의 수렴이 완료되는 것으로 확인





3. test phase

In [9]:

```
def do_test(BATCH_SIZE, INPUT_PATH, MODEL_PATH, LOG_DIR):
  startTime = time.time()
  # input generation
  with tf.Graph().as_default() as input_g:
    data = gen_data(INPUT_PATH, shuffle=False)
    n_data = len(data['png'])
    print('[%6.2f] successfully generated test data: %d samples'%(time.time()-startTime, n_data))

  # test phase
  with tf.Graph().as_default() as test_g:
    # input X: 1000 x 1000 rgb color image
    X = tf.placeholder(tf.float32, [None, 1000, 1000, 3], name='X')
    # target values Y_: 0=normal, 1=stator fault, 2=rotor fault, 3=bearing fault
    Y_ = tf.placeholder(tf.float32, [None, 4], name='Y_')

    # load inference model
    Y, cross_entropy, accuracy, incorrects = model(X, Y_)

    with tf.variable_scope('Metrics'):
      tf.summary.scalar('accuracy', accuracy)
      tf.summary.scalar('cross_entropy', cross_entropy)

    # set tensorboard summary and saver
    merged = tf.summary.merge_all()
    saver = tf.train.Saver()

    # test session
    print('----- test start -----')
    with tf.Session() as sess:
      sum_writer = tf.summary.FileWriter(LOG_DIR, sess.graph)
      tf.global_variables_initializer().run()
      saver.restore(sess, MODEL_PATH)
      avg_accuracy = 0
      for step in range(int(n_data/BATCH_SIZE)+1):
        s = step*BATCH_SIZE
        e = (step+1)*BATCH_SIZE if (step+1)*BATCH_SIZE < n_data else n_data
        if e <= s: break
        summary, acc, ent, incor, y_, y = sess.run([merged, accuracy, cross_entropy, incorrects, Y_, Y],
                                                   {X:data['X'][s:e], Y_:data['Y_'][s:e]})
        sum_writer.add_summary(summary, step+1)
        avg_accuracy += acc * (e-s)
        print('[%6.2f] step:%d, size:%d, accuracy:%f, cross entropy:%f'
              %(time.time()-startTime, step+1, e-s, acc, ent))
        if len(incor) > 0: print('   incorrects list:')
        for i in incor:
          print('   [%3d] Answer:Infer = %d:%d  at %s'
                %(s+i, tf.argmax(y_[i],0).eval(),tf.argmax(y[i],0).eval(),data['png'][s+i]))
      print('-----  test end  -----')
      print('[%6.2f] total average accuracy: %f'%(time.time()-startTime, avg_accuracy/n_data))
```

실행 예

- *INPUT*: test에 사용할 png image directory 경로 지정
- *MODEL*: test에 사용할 학습된 network parameter 경로 지정
  총 1184개 image에 대해 batch size 120, max steps 3000으로 학습된 모델 사용
- *LOG*: Tensorboard log 저장경로 지정

In [11]:

```
INPUT='/mnt/data/camus/images/20170830/'
MODEL='/mnt/data/camus/project/tmp/test/model/ckpt'
LOG='/mnt/data/camus/project/tmp/test/log/'
```

```
# do test with batch size 100 and pre-trained model parameters
do_test(100, INPUT, MODEL, LOG)
```

```
[  2.10] successfully generated test data: 200 samples
----- test start -----
INFO:tensorflow:Restoring parameters from /mnt/data/camus/project/tmp/test/model/ckpt
[  5.18] step:1, size:100, accuracy:1.000000, cross entropy:0.000251
[  6.01] step:2, size:100, accuracy:1.000000, cross entropy:0.003246
-----  test end  -----
[  6.01] total average accuracy: 1.000000
```

- Test 결과 200개 데이터에 대해 예측 정확도 **100%**