# hwalsuklee\_tensorflow-mnist-cnnmnist\_data

# Some code was borrowed from https://github.com/petewarden/tensorflow\_makefile/blob/master/tensorflow/models/image/mnist/convolutional.py  
  
from \_\_future\_\_ import absolute\_import  
from \_\_future\_\_ import division  
from \_\_future\_\_ import print\_function  
  
import gzip  
import os  
  
import numpy  
from scipy import ndimage  
  
from six.moves import urllib  
  
import tensorflow as tf  
  
SOURCE\_URL = 'http://yann.lecun.com/exdb/mnist/'  
DATA\_DIRECTORY = "data"  
  
# Params for MNIST  
IMAGE\_SIZE = 28  
NUM\_CHANNELS = 1  
PIXEL\_DEPTH = 255  
NUM\_LABELS = 10  
VALIDATION\_SIZE = 5000 # Size of the validation set.  
  
# Download MNIST data  
def maybe\_download(filename):  
 *"""Download the data from Yann's website, unless it's already here."""* if not tf.gfile.Exists(DATA\_DIRECTORY):  
 tf.gfile.MakeDirs(DATA\_DIRECTORY)  
 filepath = os.path.join(DATA\_DIRECTORY, filename)  
 if not tf.gfile.Exists(filepath):  
 filepath, \_ = urllib.request.urlretrieve(SOURCE\_URL + filename, filepath)  
 with tf.gfile.GFile(filepath) as f:  
 size = f.size()  
 print('Successfully downloaded', filename, size, 'bytes.')  
 return filepath  
  
# Extract the images  
def extract\_data(filename, num\_images):  
 *"""Extract the images into a 4D tensor [image index, y, x, channels].  
  
 Values are rescaled from [0, 255] down to [-0.5, 0.5].  
 """* print('Extracting', filename)  
 with gzip.open(filename) as bytestream:  
 bytestream.read(16)  
 buf = bytestream.read(IMAGE\_SIZE \* IMAGE\_SIZE \* num\_images \* NUM\_CHANNELS)  
 data = numpy.frombuffer(buf, dtype=numpy.uint8).astype(numpy.float32)  
 data = (data - (PIXEL\_DEPTH / 2.0)) / PIXEL\_DEPTH  
 data = data.reshape(num\_images, IMAGE\_SIZE, IMAGE\_SIZE, NUM\_CHANNELS)  
 data = numpy.reshape(data, [num\_images, -1])  
 return data  
  
# Extract the labels  
def extract\_labels(filename, num\_images):  
 *"""Extract the labels into a vector of int64 label IDs."""* print('Extracting', filename)  
 with gzip.open(filename) as bytestream:  
 bytestream.read(8)  
 buf = bytestream.read(1 \* num\_images)  
 labels = numpy.frombuffer(buf, dtype=numpy.uint8).astype(numpy.int64)  
 num\_labels\_data = len(labels)  
 one\_hot\_encoding = numpy.zeros((num\_labels\_data,NUM\_LABELS))  
 one\_hot\_encoding[numpy.arange(num\_labels\_data),labels] = 1  
 one\_hot\_encoding = numpy.reshape(one\_hot\_encoding, [-1, NUM\_LABELS])  
 return one\_hot\_encoding  
  
# Augment training data  
def expend\_training\_data(images, labels):  
  
 expanded\_images = []  
 expanded\_labels = []  
  
 j = 0 # counter  
 for x, y in zip(images, labels):  
 j = j+1  
 if j%100==0:  
 print ('expanding data : %03d / %03d' % (j,numpy.size(images,0)))  
  
 # register original data  
 expanded\_images.append(x)  
 expanded\_labels.append(y)  
  
 # get a value for the background  
 # zero is the expected value, but median() is used to estimate background's value   
 bg\_value = numpy.median(x) # this is regarded as background's value   
 image = numpy.reshape(x, (-1, 28))  
  
 for i in range(4):  
 # rotate the image with random degree  
 angle = numpy.random.randint(-15,15,1)  
 new\_img = ndimage.rotate(image,angle,reshape=False, cval=bg\_value)  
  
 # shift the image with random distance  
 shift = numpy.random.randint(-2, 2, 2)  
 new\_img\_ = ndimage.shift(new\_img,shift, cval=bg\_value)  
  
 # register new training data  
 expanded\_images.append(numpy.reshape(new\_img\_, 784))  
 expanded\_labels.append(y)  
  
 # images and labels are concatenated for random-shuffle at each epoch  
 # notice that pair of image and label should not be broken  
 expanded\_train\_total\_data = numpy.concatenate((expanded\_images, expanded\_labels), axis=1)  
 numpy.random.shuffle(expanded\_train\_total\_data)  
  
 return expanded\_train\_total\_data  
  
# Prepare MNISt data  
def prepare\_MNIST\_data(use\_data\_augmentation=True):  
 # Get the data.  
 train\_data\_filename = maybe\_download('train-images-idx3-ubyte.gz')  
 train\_labels\_filename = maybe\_download('train-labels-idx1-ubyte.gz')  
 test\_data\_filename = maybe\_download('t10k-images-idx3-ubyte.gz')  
 test\_labels\_filename = maybe\_download('t10k-labels-idx1-ubyte.gz')  
  
 # Extract it into numpy arrays.  
 train\_data = extract\_data(train\_data\_filename, 60000)  
 train\_labels = extract\_labels(train\_labels\_filename, 60000)  
 test\_data = extract\_data(test\_data\_filename, 10000)  
 test\_labels = extract\_labels(test\_labels\_filename, 10000)  
  
 # Generate a validation set.  
 validation\_data = train\_data[:VALIDATION\_SIZE, :]  
 validation\_labels = train\_labels[:VALIDATION\_SIZE,:]  
 train\_data = train\_data[VALIDATION\_SIZE:, :]  
 train\_labels = train\_labels[VALIDATION\_SIZE:,:]  
  
 # Concatenate train\_data & train\_labels for random shuffle  
 if use\_data\_augmentation:  
 train\_total\_data = expend\_training\_data(train\_data, train\_labels)  
 else:  
 train\_total\_data = numpy.concatenate((train\_data, train\_labels), axis=1)  
  
 train\_size = train\_total\_data.shape[0]  
  
 return train\_total\_data, train\_size, validation\_data, validation\_labels, test\_data, test\_labels