

```

#!/usr/bin/python
import numpy as np
import tensorflow as tf
import time
import dataset
import utilities

# Tensorflow convinience functions
def weight_variable(shape, name):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial, name=name)

def bias_variable(shape, name):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial, name=name)

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_3x3(x, name):
    return tf.nn.max_pool(x, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding='VALID', name=name)

def max_pool_2x2(x, name):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME', name=name)

def doDumbNet(trainDir, valDir, trainCsv, valCsv):
    f1 = open('log_%d' % (time.time()), 'w+')
    f1.write('AAAAA\n')
    f1.write("Start %s\n" % time.time())
    f1.flush()

    # Force CPU only mode because net overflows GPU mem
    with tf.device('/cpu:0'):
        sess = tf.Session()

        # Constants
        nClasses = 38
        imageSize = 256
        batchSize = 10
        learningRate = 1e-4
        dropOutValue = 0.5
        f1.write('nClasses: %d, imageSize: %d, batchSize: %d, learningRate: %e, dropOut: %f\n'
                % (nClasses, imageSize, batchSize, learningRate, dropOutValue))
        f1.flush()
        x = tf.placeholder("float", shape=[None, imageSize, imageSize, 1], name="Input")
        y_ = tf.placeholder("float", shape=[None, nClasses], name="Output")

        # CONVOLUTIONAL NEURAL NET
        # The first two dimensions are the patch size, the next is the number of input channels,
        # and the last is the number of output channels.
        # We will also have a bias vector with a component for each output channel.
        d1 = 32
        d2 = 32
        d3 = 64
        d4 = 64
        d5 = 64
        fc = 300

        W_conv1 = weight_variable([5, 5, 1, d1], name="Weights_conv1")
        b_conv1 = bias_variable([d1], name="b_conv1")

        # We then convolve x_image with the weight tensor,
        # add the bias, apply the ReLU function, and finally max pool.

```

```

h_conv1 = tf.nn.sigmoid(conv2d(x, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1, name="pool1")

# SECOND CONV LAYER
# In order to build a deep network, we stack several layers of this type.
# The second layer will have 64 features for each 5x5 patch.
W_conv2 = weight_variable([5, 5, d1, d2], name="Weights_conv2")
b_conv2 = bias_variable([d2], name="biases_conv2")

h_conv2 = tf.nn.sigmoid(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2, name="pool2")

# THIRD CONV LAYER
W_conv3 = weight_variable([5, 5, d2, d3], name="Weights_conv3")
b_conv3 = bias_variable([d3], name="biases_conv3")

h_conv3 = tf.nn.sigmoid(conv2d(h_pool2, W_conv3) + b_conv3)
h_pool3 = max_pool_2x2(h_conv3, name="pool3")

# FOURTH CONV LAYER
W_conv4 = weight_variable([3, 3, d3, d4], name="Weights_conv4")
b_conv4 = bias_variable([d4], name="biases_conv4")

h_conv4 = tf.nn.sigmoid(conv2d(h_pool3, W_conv4) + b_conv4)
h_pool4 = max_pool_2x2(h_conv4, name="pool4")

# FIFTH CONV LAYER
W_conv5 = weight_variable([3, 3, d4, d5], name="Weights_conv5")
b_conv5 = bias_variable([d5], name="biases_conv5")

h_conv5 = tf.nn.sigmoid(conv2d(h_pool4, W_conv5) + b_conv5)
h_pool5 = max_pool_2x2(h_conv5, name="pool5")

# DENSELY CONNECTED LAYER
# Now that the image size has been reduced to 8x8,
# we add a fully-connected layer with 300 neurons to allow processing on the entire image.
# We reshape the tensor from the pooling layer into a batch of vectors, multiply by a weight
# matrix, add a bias, and apply a ReLU.

W_fc1 = weight_variable([8 * 8 * d5, fc], name="Weights_fc1")
b_fc1 = bias_variable([fc], name="biases_fc1")

h_conv5_flat = tf.reshape(h_pool5, [-1, 8 * 8 * d5])
h_fc1 = tf.nn.sigmoid(tf.matmul(h_conv5_flat, W_fc1) + b_fc1)

# DROPOUT
keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# READOUT LAYER
W_fc2 = weight_variable([fc, nClasses], name="Weights_fc2")
b_fc2 = bias_variable([nClasses], name="biases_fc2")
y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

# Load the dataset
datasets = dataset.read_data_sets(trainDir, valDir, trainCsv, valCsv)

# Train and eval the model
cross_entropy = -tf.reduce_sum(y_ * tf.log(tf.clip_by_value(y_conv, 1e-10, 1.0)))
tf.scalar_summary('cross entropy', cross_entropy)

# train_step = tf.train.GradientDescentOptimizer(0.00001).minimize(cross_entropy)
train_step = tf.train.AdamOptimizer(learningRate).minimize(cross_entropy)

```

```

correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
sess.run(tf.initialize_all_variables())

)

# saver = tf.train.Saver()
# saver.restore(sess, 'my-model-batch1-10000')

for i in xrange(5000):
    step_start = time.time()

    batch = datasets.train.get_sequential_batch(batchSize)
    train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 1}, session=sess)

    if i % 25 == 0:
        # evaluate accuracy on random 100 samples from train set
        batch = datasets.train.get_random_batch(100)
        f1.write("step %d finished, time = %s\n" % (i, time.time() - step_start))
        acc, cross_entropyD, yD = sess.run([accuracy, cross_entropy, y_conv],
                                           feed_dict={x: batch[0], y_: batch[1], keep_prob: 1})
        f1.write("Cross entropy = " + str(cross_entropyD) + "\n")
        f1.write("Accuracy = " + str(acc) + "\n")
        f1.write("Y = " + str(np.argmax(yD, axis=1)) + "\n")
        f1.write("\n--- %s seconds ---\n\n" % (time.time() - start_time))
        f1.flush()

    f1.write("\nstep %d finished, %d seconds\n" % (i, time.time() - step_start))
    f1.flush()

saver.save(sess, 'my-model-%d' % (time.time()), global_step=10000) # Evaluate the prediction
test = datasets.validation.getAll()
acc, y_convD, correct_predictionD = sess.run([accuracy, y_conv, correct_prediction],
                                           feed_dict={x: test[0], y_: test[1], keep_prob: 1.0})

f1.write("Accuracy = " + str(acc) + "\n")
f1.write("Correct prediction %d\n" % (sum(correct_predictionD)))
f1.write("y %s\n" % str(test[1]))
f1.write("y from net %s\n" % str(np.argmax(y_convD, axis=1)))
f1.write("\n--- %s seconds ---\n\n" % (time.time() - start_time))
f1.flush()
f1.close()

start_time = time.time()
doDumbNet('train/train', 'train/validation', 'whales.csv', 'whales.csv')

```