# Appendix C: Implementation in TensorFlow

#! /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 with tf.device('/cpu:0'):

# Creates a session with log\_device\_placement set to True.

# sess = tf.Session(config=tf.ConfigProto(log\_device\_placement=True)) 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")

# There are 4 classes (labels)

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.

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")

# FORTH 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 1024 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())

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, summary\_str = sess.run([accuracy, cross\_entropy, summary\_op], # feed\_dict={x: batch[0], y\_: yTrain, keep\_prob: 1}) 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()

# summary\_writer.add\_summary(summary\_str, i) # utility.draw(h\_pool1, 113, 113, 6, 6)

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')