Wanat_Assignment_7_final

August 11, 2019

[1]:
```
# Initial deep neural network set-up from
# Geron, A. 2017. Hands-On Machine Learning with Scikit-Learn
#    & TensorFlow: Concepts, Tools, and Techniques to Build
#    Intelligent Systems. Sebastopol, Calif.: O'Reilly.
#    [ISBN-13 978-1-491-96229-9]
#    Source code available at https://github.com/ageron/handson-ml
#    See file 10_introduction_to_artificial_neural_networks.ipynb
#    Revised from MNIST to Cats and Dogs to begin Assignment 7
#    #CatsDogs# comment lines show additions/revisions for Cats and Dogs

# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports for our work
import os
import numpy as np
import tensorflow as tf
from matplotlib import pyplot as plt
# Scikit Learn for min-max scaling of the data
from sklearn.preprocessing import MinMaxScaler
# Scikit Learn for random splitting of the data
from sklearn.model_selection import train_test_split
import time

RANDOM_SEED = 9999
```

[2]:
```
# To make output stable across runs
def reset_graph(seed= RANDOM_SEED):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)
```

0.1 Preparing the data

[3]:
```
#from starter code
#CatsDogs#
# Documentation on npy binary format for saving numpy arrays for later use
#     https://towardsdatascience.com/
#             why-you-should-start-using-npy-file-more-often-df2a13cc0161
# Under the working directory, data files are in directory cats_dogs_64_128
# Read in cats and dogs grayscale 64x64 files to create training data
cats_1000_64_64_1 = np.load('cats_dogs_64-128/cats_1000_64_64_1.npy')
dogs_1000_64_64_1 = np.load('cats_dogs_64-128/dogs_1000_64_64_1.npy')
```

[4]:
```
 # for display of images
def show_grayscale_image(image):
    plt.imshow(image, cmap='gray')
    plt.axis('off')
    plt.show()
# Examine first cat and first dog grayscale images
show_grayscale_image(cats_1000_64_64_1[0,:,:,0])
show_grayscale_image(dogs_1000_64_64_1[0,:,:,0])
```

[5]:
```
height = 64
width = 64
```

[6]:
```
reset_graph()
```

[7]:
```
# Work the data for cats and dogs numpy arrays
# These numpy arrays were generated in previous data prep work
# Stack the numpy arrays for the inputs
X_cat_dog = np.concatenate((cats_1000_64_64_1, dogs_1000_64_64_1), axis = 0)
X_cat_dog = X_cat_dog.reshape(-1,width*height) # note conversion to 4096 inputs

# Scikit Learn for min-max scaling of the data
scaler = MinMaxScaler()
scaler.fit(np.array([0., 255.]).reshape(-1,1))
X_cat_dog_min_max = scaler.transform(X_cat_dog)

# Define the labels to be used 1000 cats = 0 1000 dogs = 1
y_cat_dog = np.concatenate((np.zeros((1000), dtype = np.int32),
                      np.ones((1000), dtype = np.int32)), axis = 0)

# Scikit Learn for random splitting of the data
# Random splitting of the data in to training (80%) and test (20%)
X_train, X_test, y_train, y_test = \
    train_test_split(X_cat_dog_min_max, y_cat_dog, test_size=0.20,
                     random_state = RANDOM_SEED)
```

[8]:
```
X_cat_dog_min_max.shape
```

[8]:
```
(2000, 4096)
```

[9]:
```
X_cat_dog
```

[9]:
```
array([[170., 176., 183., ...,   2.,   2.,   2.],
       [ 43.,  42.,  36., ...,  72.,  68.,  29.],
       [ 52.,  46.,  61., ...,  44.,  42.,  49.],
       ...,
       [162., 157., 142., ..., 141., 178., 117.],
       [ 79.,  80.,  81., ..., 107.,  99.,  97.],
       [139., 160., 139., ..., 220., 209., 187.]])
```

[10]:
```
print('The length of the X_train data set is: {}'.format(len(X_train)))
print('The length of the X_test data set is: {}'.format(len(X_test)))
```

```
The length of the X_train data set is: 1600
The length of the X_test data set is: 400
```

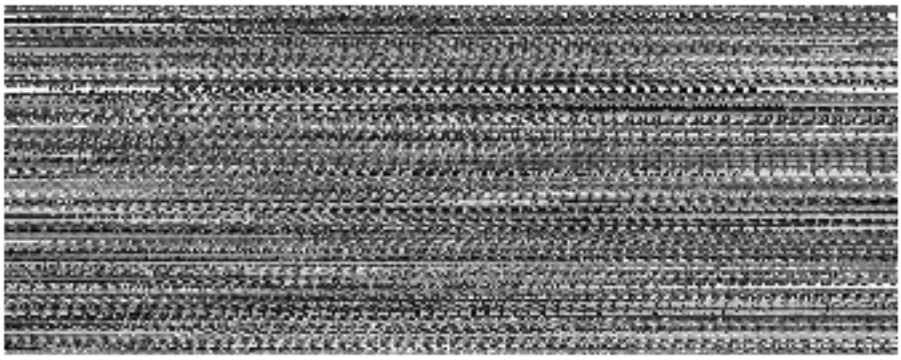
[11]:
```
y_train
```

[11]:
```
array([1, 1, 0, ..., 0, 1, 0], dtype=int32)
```

[12]:
```
X_train.shape
```

[12]:
```
(1600, 4096)
```

[13]:
```
show_grayscale_image(X_train)
```



[14]:
```
X_train[0]
```

[14]:
```
array([0.39607843, 0.40392157, 0.83137255, ..., 0.34509804, 0.20392157,
       0.21568627])
```

0.2 Set-up: Placeholders, layers, cost function, optimizer

[15]:
```
#image information
height = 64
width = 64
```

```
channels = 1  # When working with color images use channels = 3

n_inputs = height * width

#CatsDogs# Has two output values
n_outputs = 2  # binary classification for Cats and Dogs, 1 output node 0/1
```

[16]:
```
#reset graph to remove duplicate nodes
tf.reset_default_graph()

# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")
#keep_prob = tf.placeholder(tf.bool, name="keep_prob")
#keep_prob = tf.placeholder_with_default(False, shape=[], name="keep_prob")
```

[17]:
```
# create convolutional neural network layers
# https://www.tensorflow.org/tutorials/estimators/cnn

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    #X_reshaped = tf.reshape(X, shape=[-1, height, width, channels])
    #input_layer = tf.reshape(features["image"], [-1, _DEFAULT_IMAGE_SIZE,␣
 ↪_DEFAULT_IMAGE_SIZE, 3])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=3,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    pool2_flat = tf.reshape(pool2, [-1, 16 * 16 * 64])
    dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)
```

```
WARNING:tensorflow:From <ipython-input-17-8896b960ab10>:8: conv2d (from
tensorflow.python.layers.convolutional) is deprecated and will be removed in a
future version.
Instructions for updating:
Use keras.layers.conv2d instead.
WARNING:tensorflow:From /Users/jmwanat/anaconda3/envs/tf/lib/python3.7/site-
packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From <ipython-input-17-8896b960ab10>:9: max_pooling2d (from
```

```
tensorflow.python.layers.pooling) is deprecated and will be removed in a future
version.
Instructions for updating:
Use keras.layers.max_pooling2d instead.
WARNING:tensorflow:From <ipython-input-17-8896b960ab10>:13: dense (from
tensorflow.python.layers.core) is deprecated and will be removed in a future
version.
Instructions for updating:
Use keras.layers.dense instead.
WARNING:tensorflow:From <ipython-input-17-8896b960ab10>:14: dropout (from
tensorflow.python.layers.core) is deprecated and will be removed in a future
version.
Instructions for updating:
Use keras.layers.dropout instead.
```

[18]:
```
# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)
```

[18]:
```
<tf.Tensor 'loss_1:0' shape=() dtype=string>
```

[19]:
```
# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)
```

[20]:
```
# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[21]:
```
# Need to shuffle X and y data sets or the model does not train on all images
# if you don't do this the model is very inaccurate because it is not looking
# at all the images
# Used function defined by Geron 2017
#https://github.com/ageron/handson-ml/blob/master/
 ↪10_introduction_to_artificial_neural_networks.ipynb
def shuffle_batch(X, y, batch_size):
    rnd_idx = np.random.permutation(len(X))
    n_batches = len(X) // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch, y_batch = X[batch_idx], y[batch_idx]
```

```
        yield X_batch, y_batch
```

0.3 Model with two convolutional layers and kernel size = 3

[22]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time1 = total_time
```

```
Epoch: 0, Loss: 0.6930983066558838, Accuracy: 0.5
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.3935135006904602, Accuracy: 0.8199999928474426
Training pass: 5
Training pass: 6
Training pass: 7
```

```
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.10882680118083954, Accuracy: 0.9599999785423279
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.0029092964250594378, Accuracy: 1.0
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.0016163305845111609, Accuracy: 1.0
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.0006219355273060501, Accuracy: 1.0
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.000662698526866734, Accuracy: 1.0
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.000490290520247072, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.675000011920929
Training is complete!
Time to train and evaluate model: 246.689 seconds
```

[38]:
```
#pass number two
#reset graph to remove duplicate nodes
tf.reset_default_graph()

# create placeholders for variables
```

```
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=3,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    pool2_flat = tf.reshape(pool2, [-1, 16 * 16 * 64])
    dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[39]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
```

```
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy1b = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy1b))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy1b = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy1b))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time1b = total_time
```

```
Epoch: 0, Loss: 0.6878699660301208, Accuracy: 0.699999988079071
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.30466169118881226, Accuracy: 0.8799999952316284
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.0999460443854332, Accuracy: 0.9800000190734863
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.0053083635866642, Accuracy: 1.0
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
```

```
Epoch: 20, Loss: 0.000864103902131319, Accuracy: 1.0
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.00041343815973959863, Accuracy: 1.0
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.00024148965894710273, Accuracy: 1.0
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.0002277550520375371, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.6499999761581421
Training is complete!
Time to train and evaluate model: 262.337 seconds
```

[40]:
```
#pass number three
#reset graph to remove duplicate nodes
tf.reset_default_graph()

# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=3,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    pool2_flat = tf.reshape(pool2, [-1, 16 * 16 * 64])
    dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.
 ↪relu)
```

```
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[41]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy1c = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy1c))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy1c = accuracy.eval(feed_dict={X: X_test, y: y_test})
```

```
    print("Test accuracy: {}".format(test_accuracy1c))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time1c = total_time
```

```
Epoch: 0, Loss: 0.691304624080658, Accuracy: 0.6000000238418579
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.39895880222320557, Accuracy: 0.8399999737739563
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.058477021753787994, Accuracy: 1.0
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.003173459554091096, Accuracy: 1.0
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.0010574075859040022, Accuracy: 1.0
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.00039286629180423915, Accuracy: 1.0
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.0004557911306619644, Accuracy: 1.0
Training pass: 30
```

```
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.00010838323942152783, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.6349999904632568
Training is complete!
Time to train and evaluate model: 261.386 seconds
```

0.4 Model with two convolutional layers and kernel size = 5

[23]:
```
#reset graph to remove duplicate nodes
tf.reset_default_graph()
```

[24]:
```
# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=5,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    pool2_flat = tf.reshape(pool2, [-1, 16 * 16 * 64])
    dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
```

```
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[25]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy2 = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy2))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy2 = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy2))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time2 = total_time
```

```
Epoch: 0, Loss: 0.6826713681221008, Accuracy: 0.5600000023841858
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
```

```
Training pass: 4
Epoch: 5, Loss: 0.504966139793396, Accuracy: 0.7799999713897705
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.14913791418075562, Accuracy: 0.9399999976158142
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.0078191002830863, Accuracy: 1.0
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.0013011860428377986, Accuracy: 1.0
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.0007135512423701584, Accuracy: 1.0
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.0001431697455700487, Accuracy: 1.0
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 9.693973697721958e-05, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.6274999976158142
Training is complete!
Time to train and evaluate model: 315.320 seconds
```

[42]:
```
# pass number 2
#reset graph to remove duplicate nodes
tf.reset_default_graph()

# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=5,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    pool2_flat = tf.reshape(pool2, [-1, 16 * 16 * 64])
    dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[43]:
```
n_epochs = 40
batch_size = 50
```

```
# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy2b = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy2b))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy2b = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy2b))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time2b = total_time
```

```
Epoch: 0, Loss: 0.6874706149101257, Accuracy: 0.6399999856948853
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.476093590259552, Accuracy: 0.7200000286102295
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.027401035651564598, Accuracy: 1.0
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.000981886056251824, Accuracy: 1.0
```

```
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.00029400616767816246, Accuracy: 1.0
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.00014796639152336866, Accuracy: 1.0
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.0001676727697486058, Accuracy: 1.0
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 5.4721102060284466e-05, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.6549999713897705
Training is complete!
Time to train and evaluate model: 329.350 seconds
```

[44]:
```python
# pass number 3
#reset graph to remove duplicate nodes
tf.reset_default_graph()

# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=5,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
```

```
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    pool2_flat = tf.reshape(pool2, [-1, 16 * 16 * 64])
    dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[45]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy2c = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
```

```
            print("Epoch: {}, Loss: {}, Accuracy: {}".
↪format(epoch,train_loss,train_accuracy2c))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy2c = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy2c))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time2c = total_time
```

```
Epoch: 0, Loss: 0.6942018866539001, Accuracy: 0.4399999976158142
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.6022956967353821, Accuracy: 0.6600000262260437
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.1062474250793457, Accuracy: 0.9800000190734863
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.002700437093153596, Accuracy: 1.0
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.0008973446674644947, Accuracy: 1.0
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.00026652656379155815, Accuracy: 1.0
Training pass: 25
```

```
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.0002040900435531512, Accuracy: 1.0
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.00013656083319801837, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.6025000214576721
Training is complete!
Time to train and evaluate model: 323.162 seconds
```

0.5 Model with 5 layers and kernel size = 3

[26]:
```
#reset graph to remove duplicate nodes
tf.reset_default_graph()
```

[27]:
```
# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=3,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    conv3 = tf.layers.conv2d(pool2, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    conv4 = tf.layers.conv2d(conv3, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    conv5 = tf.layers.conv2d(conv4, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2], strides=2)
    pool3_flat = tf.reshape(pool3, [-1, 8 * 8 * 64])
    dense = tf.layers.dense(inputs=pool3_flat, units=1024, activation=tf.nn.
 ↪relu)
```

```
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[28]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy3 = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy3))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy3 = accuracy.eval(feed_dict={X: X_test, y: y_test})
```

```
    print("Test accuracy: {}".format(test_accuracy3))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time3 = total_time
```

```
Epoch: 0, Loss: 0.6974614858627319, Accuracy: 0.3400000035762787
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.37139636278152466, Accuracy: 0.800000011920929
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.1726275086402893, Accuracy: 0.9399999976158142
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.06924143433570862, Accuracy: 0.9800000190734863
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.001855303067713976, Accuracy: 1.0
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.0005604480393230915, Accuracy: 1.0
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.00025734055088832974, Accuracy: 1.0
Training pass: 30
```

```
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.0001694101665634662, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.7024999856948853
Training is complete!
Time to train and evaluate model: 217.400 seconds
```

[46]:
```
#pass number 2
#reset graph to remove duplicate nodes
tf.reset_default_graph()

# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=3,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    conv3 = tf.layers.conv2d(pool2, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    conv4 = tf.layers.conv2d(conv3, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    conv5 = tf.layers.conv2d(conv4, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2], strides=2)
    pool3_flat = tf.reshape(pool3, [-1, 8 * 8 * 64])
    dense = tf.layers.dense(inputs=pool3_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
```

```
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[47]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy3b = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy3b))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy3b = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy3b))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0
```

```
time3b = total_time
```

```
Epoch: 0, Loss: 0.680797278881073, Accuracy: 0.5400000214576721
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.3845057189464569, Accuracy: 0.8399999737739563
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.1837325096130371, Accuracy: 0.9599999785423279
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.03914757817983627, Accuracy: 1.0
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.0012174402363598347, Accuracy: 1.0
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.0005992799415253103, Accuracy: 1.0
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.0002747261605691165, Accuracy: 1.0
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.0001236234966199845, Accuracy: 1.0
Training pass: 35
Training pass: 36
```

```
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.6575000286102295
Training is complete!
Time to train and evaluate model: 214.717 seconds
```

[48]:
```
#pass number 3
#reset graph to remove duplicate nodes
tf.reset_default_graph()

# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=3,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    conv3 = tf.layers.conv2d(pool2, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    conv4 = tf.layers.conv2d(conv3, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    conv5 = tf.layers.conv2d(conv4, filters=64 , kernel_size=3, padding="same",␣
 ↪activation=tf.nn.relu)
    pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2], strides=2)
    pool3_flat = tf.reshape(pool3, [-1, 8 * 8 * 64])
    dense = tf.layers.dense(inputs=pool3_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)
```

```
# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[49]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy3c = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy3c))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy3c = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy3c))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time3c = total_time
```

```
Epoch: 0, Loss: 0.6954905986785889, Accuracy: 0.4000000059604645
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
```

```
Epoch: 5, Loss: 0.6246107816696167, Accuracy: 0.6399999856948853
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.23123756051063538, Accuracy: 0.9399999976158142
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.1272122710943222, Accuracy: 0.9599999785423279
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.027766089886426926, Accuracy: 0.9800000190734863
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.0009274613694287837, Accuracy: 1.0
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.0006219672504812479, Accuracy: 1.0
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.0002745779638644308, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.6474999785423279
Training is complete!
Time to train and evaluate model: 214.786 seconds
```

0.6 Model with 5 layers and kernel size = 5

[29]:
```
#reset graph to remove duplicate nodes
tf.reset_default_graph()
```

[30]:
```
# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=5,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    conv3 = tf.layers.conv2d(pool2, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    conv4 = tf.layers.conv2d(conv3, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    conv5 = tf.layers.conv2d(conv4, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    pool3 = tf.layers.max_pooling2d(inputs=conv5, pool_size=[2, 2], strides=2)
    pool3_flat = tf.reshape(pool3, [-1, 8 * 8 * 64])
    dense = tf.layers.dense(inputs=pool3_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)
```

```
# save trained model parameters to disk
saver = tf.train.Saver()
```

[31]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy4 = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy4))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy4 = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy4))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time4 = total_time
```

```
Epoch: 0, Loss: 0.6906610727310181, Accuracy: 0.6000000238418579
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.6932476758956909, Accuracy: 0.46000000834465027
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
```

```
Epoch: 10, Loss: 0.6710488796234131, Accuracy: 0.6000000238418579
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.6457678079605103, Accuracy: 0.6200000047683716
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.4887639582157135, Accuracy: 0.7599999904632568
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.40546199679374695, Accuracy: 0.8399999737739563
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.10129057615995407, Accuracy: 0.9599999785423279
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.07476308196783066, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.6499999761581421
Training is complete!
Time to train and evaluate model: 457.478 seconds
```

[50]:
```
# pass number 2
#reset graph to remove duplicate nodes
tf.reset_default_graph()

# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")
```

```
with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=5,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    conv3 = tf.layers.conv2d(pool2, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    conv4 = tf.layers.conv2d(conv3, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    conv5 = tf.layers.conv2d(conv4, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    pool3 = tf.layers.max_pooling2d(inputs=conv5, pool_size=[2, 2], strides=2)
    pool3_flat = tf.reshape(pool3, [-1, 8 * 8 * 64])
    dense = tf.layers.dense(inputs=pool3_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[51]:
```
n_epochs = 40
batch_size = 50
```

```
# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy4b = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy4b))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy4b = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy4b))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time4b = total_time
```

```
Epoch: 0, Loss: 0.6979684233665466, Accuracy: 0.3799999952316284
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.6938040852546692, Accuracy: 0.46000000834465027
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.6691261529922485, Accuracy: 0.5799999833106995
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.5359758138656616, Accuracy: 0.7400000095367432
```

```
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.37325337529182434, Accuracy: 0.8399999737739563
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.2165774554014206, Accuracy: 0.9200000166893005
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.08664220571517944, Accuracy: 0.9599999785423279
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.032271381467580795, Accuracy: 1.0
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.6949999928474426
Training is complete!
Time to train and evaluate model: 458.938 seconds
```

[52]:
```
# pass number 3
#reset graph to remove duplicate nodes
tf.reset_default_graph()

# create placeholders for variables
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")

with tf.name_scope("cnn"):
    input_layer = tf.reshape(X, shape=[-1, height, width, channels])
    conv1 = tf.layers.conv2d(input_layer, filters=32 , kernel_size=5,␣
 ↪padding="same", activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
```

```
    conv2 = tf.layers.conv2d(pool1, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
    conv3 = tf.layers.conv2d(pool2, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    conv4 = tf.layers.conv2d(conv3, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    conv5 = tf.layers.conv2d(conv4, filters=64 , kernel_size=5, padding="same",␣
 ↪activation=tf.nn.relu)
    pool3 = tf.layers.max_pooling2d(inputs=conv5, pool_size=[2, 2], strides=2)
    pool3_flat = tf.reshape(pool3, [-1, 8 * 8 * 64])
    dense = tf.layers.dense(inputs=pool3_flat, units=1024, activation=tf.nn.
 ↪relu)
    dense = tf.layers.dropout(inputs=dense, rate=0.5)
    logits = tf.layers.dense(inputs=dense, units=2)

# define cost function
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,␣
 ↪logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
tf.summary.scalar('xentropy', xentropy)
tf.summary.scalar('loss', loss)

# use optimizer to modify parameters to minimize cost function
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer().minimize(loss)

# evaluate the model using accuracy
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# save trained model parameters to disk
saver = tf.train.Saver()
```

[53]:
```
n_epochs = 40
batch_size = 50


# Initialize a session so that we can run TensorFlow operations
with tf.Session() as session:
    t0 = time.time()
# Run the global variable initializer to initialize all variables and layers of␣
 ↪the neural network
    session.run(tf.global_variables_initializer())
```

```
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
# Feed in the training data and do one step of neural network training
            session.run(optimizer, feed_dict={X: X_batch, y: y_batch})
        if epoch % 5 == 0:
            train_loss = session.run(loss, feed_dict={X: X_batch, y:y_batch})
            train_accuracy4c = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            print("Epoch: {}, Loss: {}, Accuracy: {}".
 ↪format(epoch,train_loss,train_accuracy4c))

        # Print the current training status to the screen
        print("Training pass: {}".format(epoch))
    test_accuracy4c = accuracy.eval(feed_dict={X: X_test, y: y_test})
    print("Test accuracy: {}".format(test_accuracy4c))
    # Training is now complete!
    print("Training is complete!")
    t1 = time.time()
    print("Time to train and evaluate model: {:.3f} seconds".format(t1-t0))
    total_time = t1 - t0

time4c = total_time
```

```
Epoch: 0, Loss: 0.6931596994400024, Accuracy: 0.5
Training pass: 0
Training pass: 1
Training pass: 2
Training pass: 3
Training pass: 4
Epoch: 5, Loss: 0.6919700503349304, Accuracy: 0.6000000238418579
Training pass: 5
Training pass: 6
Training pass: 7
Training pass: 8
Training pass: 9
Epoch: 10, Loss: 0.6939313411712646, Accuracy: 0.46000000834465027
Training pass: 10
Training pass: 11
Training pass: 12
Training pass: 13
Training pass: 14
Epoch: 15, Loss: 0.6928460597991943, Accuracy: 0.5199999809265137
Training pass: 15
Training pass: 16
Training pass: 17
Training pass: 18
Training pass: 19
Epoch: 20, Loss: 0.6911152005195618, Accuracy: 0.6000000238418579
```

```
Training pass: 20
Training pass: 21
Training pass: 22
Training pass: 23
Training pass: 24
Epoch: 25, Loss: 0.6925461292266846, Accuracy: 0.5400000214576721
Training pass: 25
Training pass: 26
Training pass: 27
Training pass: 28
Training pass: 29
Epoch: 30, Loss: 0.6927719116210938, Accuracy: 0.5199999809265137
Training pass: 30
Training pass: 31
Training pass: 32
Training pass: 33
Training pass: 34
Epoch: 35, Loss: 0.6935951709747314, Accuracy: 0.47999998927116394
Training pass: 35
Training pass: 36
Training pass: 37
Training pass: 38
Training pass: 39
Test accuracy: 0.47999998927116394
Training is complete!
Time to train and evaluate model: 458.579 seconds
```

0.7 Summary of model performance with Tensorflow

[59]:
```
summary_models = {
    'Number of Convolutional Layers' : [2, 2, 5, 5],
    'Kernel size' : [3, 5, 3, 5],
    'Processing Time' : [round(time1, 3), round(time2,3), round(time3,3),␣
 ↪round(time4,3)],
    'Processing Time #2' : [round(time1b, 3), round(time2b,3), round(time3b,3),␣
 ↪round(time4b,3)],
    'Processing Time #3' : [round(time1c, 3), round(time2c,3), round(time3c,3),␣
 ↪round(time4c,3)],
 #   'Train Accuracy' : [round(train_accuracy, 3), round(train_accuracy2, 3),␣
 ↪round(train_accuracy3, 3), round(train_accuracy4,3)],
    'Test Accruacy' : [round(test_accuracy, 3), round(test_accuracy2, 3),␣
 ↪round(test_accuracy3, 3), round(test_accuracy4, 3)],
    'Test Accruacy #2' : [round(test_accuracy1b, 3), round(test_accuracy2b, 3),␣
 ↪round(test_accuracy3b, 3), round(test_accuracy4b, 3)],
    'Test Accruacy #3' : [round(test_accuracy1c, 3), round(test_accuracy2c, 3),␣
 ↪round(test_accuracy3c, 3), round(test_accuracy4c, 3)]
}
```

[60]:
```
import pandas as pd
summary_models_df = pd.DataFrame(summary_models)
summary_models_df
```

[60]:

	Number of Convolutional Layers	Kernel size	Processing Time	Processing Time #2	Processing Time #3	Test Accruacy	Test Accruacy #2	Test Accruacy #3
0	2	3	246.689	262.337	261.386	0.675	0.650	0.635
1	2	5	315.320	329.350	323.162	0.628	0.655	0.602
2	5	3	217.400	214.717	214.786	0.702	0.658	0.648
3	5	5	457.478	458.938	458.579	0.650	0.695	0.480

[62]:
```
test_avg1 = (test_accuracy + test_accuracy1b + test_accuracy1c) / 3
test_avg2 = (test_accuracy2 + test_accuracy2b + test_accuracy2c) / 3
test_avg3 = (test_accuracy3 + test_accuracy3b + test_accuracy3c) / 3
test_avg4 = (test_accuracy4 + test_accuracy4b + test_accuracy4c) / 3

model_average = {
    'Number of Convolutional Layers' : [2, 2, 5, 5],
```

```
    'Kernel size' : [3, 5, 3, 5],
    'Test Accruacy Average' : [round(test_avg1, 3), round(test_avg2, 3),␣
 ↪round(test_avg3, 3), round(test_avg4, 3)],
}

model_average_df = pd.DataFrame(model_average)
model_average_df
```

[62]:

	Number of Convolutional Layers	Kernel size	Test Accruacy Average
0	2	3	0.653
1	2	5	0.628
2	5	3	0.669
3	5	5	0.608

0.8 Summary of other models utilizing various parameters

Various parameters were assessed with the model configurations as indicated. The test accuracy reflects only one assessment.

[ ]:
```
#My other attempts summarized below.
# 3 conv layers
#learning rate 1 e-4
#Gradient optimizer
#with dropout 0.5
#Test accuracy: 0.5649999976158142
#Training is complete!
#Time to train and evaluate model: 200.621 seconds

# 3 conv layers
#learning rate 1 e-2
#gradient optimizer
# with dropout 0.5
#Test accuracy: 0.6600000262260437
#Training is complete!
#Time to train and evaluate model: 199.935 seconds

# 3 conv layers
#adam optimizer
# no dropout
#Test accuracy: 0.6449999809265137
#Training is complete!
#Time to train and evaluate model: 232.776 seconds

# 3 conv layers
#adam optimizer
# with dropout
#Test accuracy: 0.6600000262260437
#Training is complete!
```

```
#Time to train and evaluate model: 233.716 seconds


# 5 conv layers
# learning rate 0.1
# gradient optimizer
# with dropout
#Test accuracy: 0.5199999809265137
#Training is complete!
#Time to train and evaluate model: 252.955 seconds


# 5 conv layers 64
# learning rate 0.01
# gradient optimizer
# with dropout

#Test accuracy: 0.6424999833106995
#Training is complete!
#Time to train and evaluate model: 252.684 seconds


# 5 conv layers 128
# learning rate 0.01
# gradient optimizer
# with dropout
#Test accuracy: 0.6399999856948853
#Training is complete!
#Time to train and evaluate model: 436.043 seconds
```

0.9 Keras

[63]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
from tensorflow.python.keras.callbacks import TensorBoard
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, Reshape
```

```
Using TensorFlow backend.
```

[64]:
```
#reset graph to remove duplicate nodes
tf.reset_default_graph()

model = Sequential()

model.add(Conv2D(16, (3, 3), padding='same', input_shape=(64,64,1)))
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
#model.add(Dense(1))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
WARNING:tensorflow:From /Users/jmwanat/anaconda3/envs/tf/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
```

[65]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 16)        160
_________________________________________________________________
activation_1 (Activation)    (None, 64, 64, 16)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        4640
_________________________________________________________________
activation_2 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 64)        18496
```

```
_________________________________________________________________
activation_3 (Activation)    (None, 13, 13, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
=================================================================
Total params: 1,204,482
Trainable params: 1,204,482
Non-trainable params: 0
_________________________________________________________________
```

[66]:
```
X_train_reshape = tf.reshape(X_train, shape=[-1, height, width, channels])
X_test_reshape = tf.reshape(X_test, shape=[-1, height, width, channels])



t0 = time.time()
# Train the model, iterating on the data in batches of 50 samples
history = model.fit(X_train_reshape, y_train, epochs=7, steps_per_epoch=50)
# Evaluate the model
score = model.evaluate(X_test_reshape, y_test, steps=50)
t1 = time.time()
print('Total time to train and evaluate model: {:.3f}'.format(t1-t0))
time_keras1 = t1-t0
```

```
WARNING:tensorflow:From /Users/jmwanat/anaconda3/envs/tf/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/7
50/50 [==============================] - 65s 1s/step - loss: 0.6196 - acc:
0.6438
Epoch 2/7
50/50 [==============================] - 61s 1s/step - loss: 0.4055 - acc:
0.8222
Epoch 3/7
```

```
50/50 [==============================] - 61s 1s/step - loss: 0.2040 - acc:
0.9309
Epoch 4/7
50/50 [==============================] - 62s 1s/step - loss: 0.1180 - acc:
0.9637
Epoch 5/7
50/50 [==============================] - 63s 1s/step - loss: 0.0405 - acc:
0.9947
Epoch 6/7
50/50 [==============================] - 63s 1s/step - loss: 0.0183 - acc:
0.9992
Epoch 7/7
50/50 [==============================] - 63s 1s/step - loss: 0.0095 - acc:
0.9998
50/50 [==============================] - 5s 109ms/step
Total time to train and evaluate model: 442.789
```

[68]:
```
loss_keras1, accuracy_keras1 = score
```

[77]:
```
# pass number 2
#reset graph to remove duplicate nodes
tf.reset_default_graph()
# define function re-initialize model weights
#https://www.codementor.io/nitinsurya/
 ↪how-to-re-initialize-keras-model-weights-et41zre2g
#def reset_weights(model):
#    session = tf.keras.backend.get_session()
#    for layer in model.layers:
#        if hasattr(layer, 'kernel_initializer'):
#            layer.kernel.initializer.run(session=session)
#reset_weights(model)

# Clean up the TF session
import keras
keras.backend.clear_session()

model = Sequential()

model.add(Conv2D(16, (3, 3), padding='same', input_shape=(64,64,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
#model.add(Dense(1))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[78]:
```
X_train_reshape = tf.reshape(X_train, shape=[-1, height, width, channels])
X_test_reshape = tf.reshape(X_test, shape=[-1, height, width, channels])



t0 = time.time()
# Train the model, iterating on the data in batches of 50 samples
history2 = model.fit(X_train_reshape, y_train, epochs=7, steps_per_epoch=50)
# Evaluate the model
score2 = model.evaluate(X_test_reshape, y_test, steps=50)
t1 = time.time()
print('Total time to train and evaluate model: {:.3f}'.format(t1-t0))
time_keras2 = t1-t0
```

```
Epoch 1/7
50/50 [==============================] - 64s 1s/step - loss: 0.6470 - acc:
0.6123
Epoch 2/7
50/50 [==============================] - 62s 1s/step - loss: 0.4522 - acc:
0.7866
Epoch 3/7
50/50 [==============================] - 62s 1s/step - loss: 0.2174 - acc:
0.9223
Epoch 4/7
50/50 [==============================] - 62s 1s/step - loss: 0.0662 - acc:
0.9876
Epoch 5/7
50/50 [==============================] - 62s 1s/step - loss: 0.0182 - acc:
0.9992
Epoch 6/7
50/50 [==============================] - 62s 1s/step - loss: 0.0070 - acc:
0.9999
Epoch 7/7
50/50 [==============================] - 62s 1s/step - loss: 0.0036 - acc:
1.0000
```

```
50/50 [==============================] - 5s 105ms/step
Total time to train and evaluate model: 441.637
```

[80]:
```
loss_keras2, accuracy_keras2 = score2
```

[81]:
```
# pass number 3
#reset graph to remove duplicate nodes
tf.reset_default_graph()
# define function re-initialize model weights
#https://www.codementor.io/nitinsurya/
 ↪how-to-re-initialize-keras-model-weights-et41zre2g
#def reset_weights(model):
#    session = tf.keras.backend.get_session()
#    for layer in model.layers:
#        if hasattr(layer, 'kernel_initializer'):
#            layer.kernel.initializer.run(session=session)
#reset_weights(model)

# Clean up the TF session
#import keras
keras.backend.clear_session()

model = Sequential()

model.add(Conv2D(16, (3, 3), padding='same', input_shape=(64,64,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
#model.add(Dense(1))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[82]:
```
X_train_reshape = tf.reshape(X_train, shape=[-1, height, width, channels])
X_test_reshape = tf.reshape(X_test, shape=[-1, height, width, channels])



t0 = time.time()
# Train the model, iterating on the data in batches of 50 samples
history3 = model.fit(X_train_reshape, y_train, epochs=7, steps_per_epoch=50)
# Evaluate the model
score3 = model.evaluate(X_test_reshape, y_test, steps=50)
t1 = time.time()
print('Total time to train and evaluate model: {:.3f}'.format(t1-t0))
time_keras3 = t1-t0
```

```
Epoch 1/7
50/50 [==============================] - 64s 1s/step - loss: 0.5776 - acc:
0.6878
Epoch 2/7
50/50 [==============================] - 61s 1s/step - loss: 0.3130 - acc:
0.8706
Epoch 3/7
50/50 [==============================] - 61s 1s/step - loss: 0.1058 - acc:
0.9729
Epoch 4/7
50/50 [==============================] - 62s 1s/step - loss: 0.0245 - acc:
0.9984
Epoch 5/7
50/50 [==============================] - 62s 1s/step - loss: 0.0075 - acc:
0.9999
Epoch 6/7
50/50 [==============================] - 62s 1s/step - loss: 0.0035 - acc:
1.0000
Epoch 7/7
50/50 [==============================] - 62s 1s/step - loss: 0.0020 - acc:
1.0000
50/50 [==============================] - 5s 105ms/step
Total time to train and evaluate model: 438.650
```

[83]:
```
loss_keras3, accuracy_keras3 = score3
```

[84]:
```
keras_summary_models = {
    'Processing Time' : [round(time_keras1, 3), round(time_keras2,3),␣
 ↪round(time_keras3,3)],
    'Loss' : [round(loss_keras1, 3), round(loss_keras2, 3), round(loss_keras3,␣
 ↪3)],
    'Test Accruacy' : [round(accuracy_keras1, 3), round(accuracy_keras2, 3),␣
 ↪round(accuracy_keras3, 3)]
```

```
}

keras_summary_models_df = pd.DataFrame(keras_summary_models)
keras_summary_models_df
```

[84]:

	Processing Time	Loss	Test Accruacy
0	442.789	1.259	0.70
1	441.637	1.650	0.68
2	438.650	1.808	0.67

waiting_for_network_to_train.jpeg

[ ]:
```
#source for picture:
#https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/
```