In [1]:
```
# load train and test dataset
#load dataset
import gzip
import sys
import pickle
import numpy as np

#load dataset
def load_dataset():
    f = gzip.open('mnist.pkl.gz', 'rb')
    if sys.version_info < (3,):
        data = pickle.load(f)
    else:
        data = pickle.load(f, encoding='bytes')
    f.close()
    (trainX,trainY ), (testX, testY) = data
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

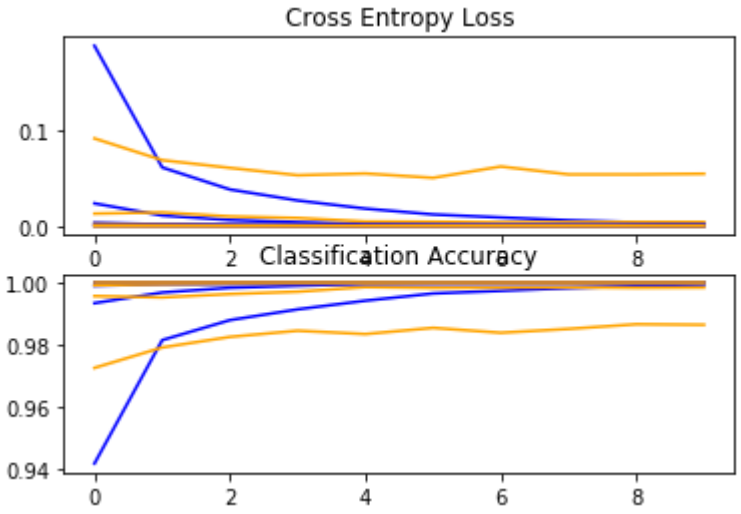
In [2]:
```
# cnn model for mnist
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD

# load train and test dataset
#load dataset
import gzip
import sys
import pickle
import numpy as np

#load dataset
def load_dataset():
    f = gzip.open('mnist.pkl.gz', 'rb')
    if sys.version_info < (3,):
        data = pickle.load(f)
    else:
        data = pickle.load(f, encoding='bytes')
    f.close()
    (trainX,trainY ), (testX, testY) = data
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm

# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uni
form', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
```

```
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['ac
curacy'])
    return model

# evaluate a model using k-fold cross-validation
def evaluate_model(model, dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX
[test_ix], dataY[test_ix]
        # fit model
        #In general, batch size of 32 is a good starting point, and we can try
with 64, 128, and 256.
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validati
on_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories

# plot diagnostic learning curves
def summarize_diagnostics(histories):
    for i in range(len(histories)):
        # plot loss
        pyplot.subplot(211)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(histories[i].history['loss'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_loss'], color='orange', label='t
est')
        # plot accuracy
        pyplot.subplot(212)
        pyplot.title('Classification Accuracy')
        pyplot.plot(histories[i].history['acc'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_acc'], color='orange', label='te
st')
    pyplot.show()

# summarize model performance
def summarize_performance(scores):
    # print summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores
)*100, len(scores)))
    # box and whisker plots of results
    pyplot.boxplot(scores)
    pyplot.show()
```

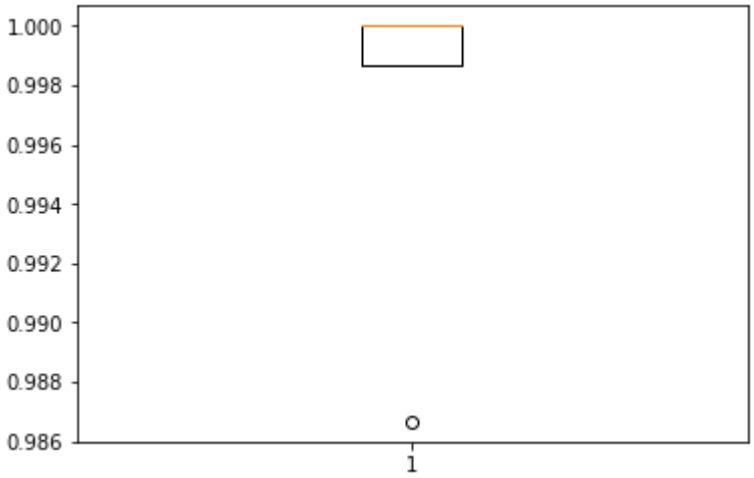
```
Using TensorFlow backend.
```

In [3]:
```
# for evaluating a model
# load dataset
trainX, trainY, testX, testY = load_dataset()
# prepare pixel data
trainX, testX = prep_pixels(trainX, testX)
# define model
model = define_model()
# evaluate model
scores, histories = evaluate_model(model, trainX, trainY)
# learning curves
summarize_diagnostics(histories)
# summarize estimated performance
summarize_performance(scores)
```

```
WARNING:tensorflow:From C:\Users\code-tech\Anaconda3\lib\site-packages\tensor
flow\python\framework\op_def_library.py:263: colocate_with (from tensorflow.p
ython.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\code-tech\Anaconda3\lib\site-packages\tensor
flow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_o
ps) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
> 98.667
> 99.867
> 100.000
> 100.000
> 100.000

C:\Users\code-tech\Anaconda3\lib\site-packages\matplotlib\figure.py:98: Matpl
otlibDeprecationWarning:
Adding an axes using the same arguments as a previous axes currently reuses t
he earlier instance.  In a future version, a new instance will always be crea
ted and returned.  Meanwhile, this warning can be suppressed, and the future
behavior ensured, by passing a unique label to each axes instance.
  "Adding an axes using the same arguments as a previous axes "
```




```
Accuracy: mean=99.707 std=0.523, n=5
```



In [6]:
```
# fit model

model.fit(trainX, trainY, epochs=10, batch_size=200, verbose=0)
print("model completed")
# save model
model.save('digit3_model.h5')
```

```
model completed
```

In [7]:
```
# evaluate the deep model on the test dataset
# make a prediction for a new image.
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # load model
    model = load_model('digit3_model.h5')
    # evaluate model on test dataset
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('Accuracy of CNN > %.3f' % (acc * 100.0))

# entry point, run the test harness
run_test_harness()
```

```
Accuracy of CNN > 98.730
```

In [8]:
```
# make a prediction for a new image.
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model

# load and prepare the image
def load_image(filename):
    # load the image
    img = load_img(filename, grayscale=True, target_size=(28, 28))
    # convert to array
    img = img_to_array(img)
    # reshape into a single sample with 1 channel
    img = img.reshape(1,28, 28,1)
    # prepare pixel data
    img = img.astype('float32')
    img = img / 255.0
    return img

# load an image and predict the class
def run_example():
    # load the image
    img = load_image('sample_image.png')
    # load model
    model = load_model('digit3_model.h5')
    # predict the class
    digit = model.predict_classes(img)
    print("Image digit is ",digit[0])

# entry point, run the example
run_example()
```

```
C:\Users\code-tech\Anaconda3\lib\site-packages\keras_preprocessing\image\util
s.py:98: UserWarning: grayscale is deprecated. Please use color_mode = "grays
cale"
  warnings.warn('grayscale is deprecated. Please use '

Image digit is  7
```