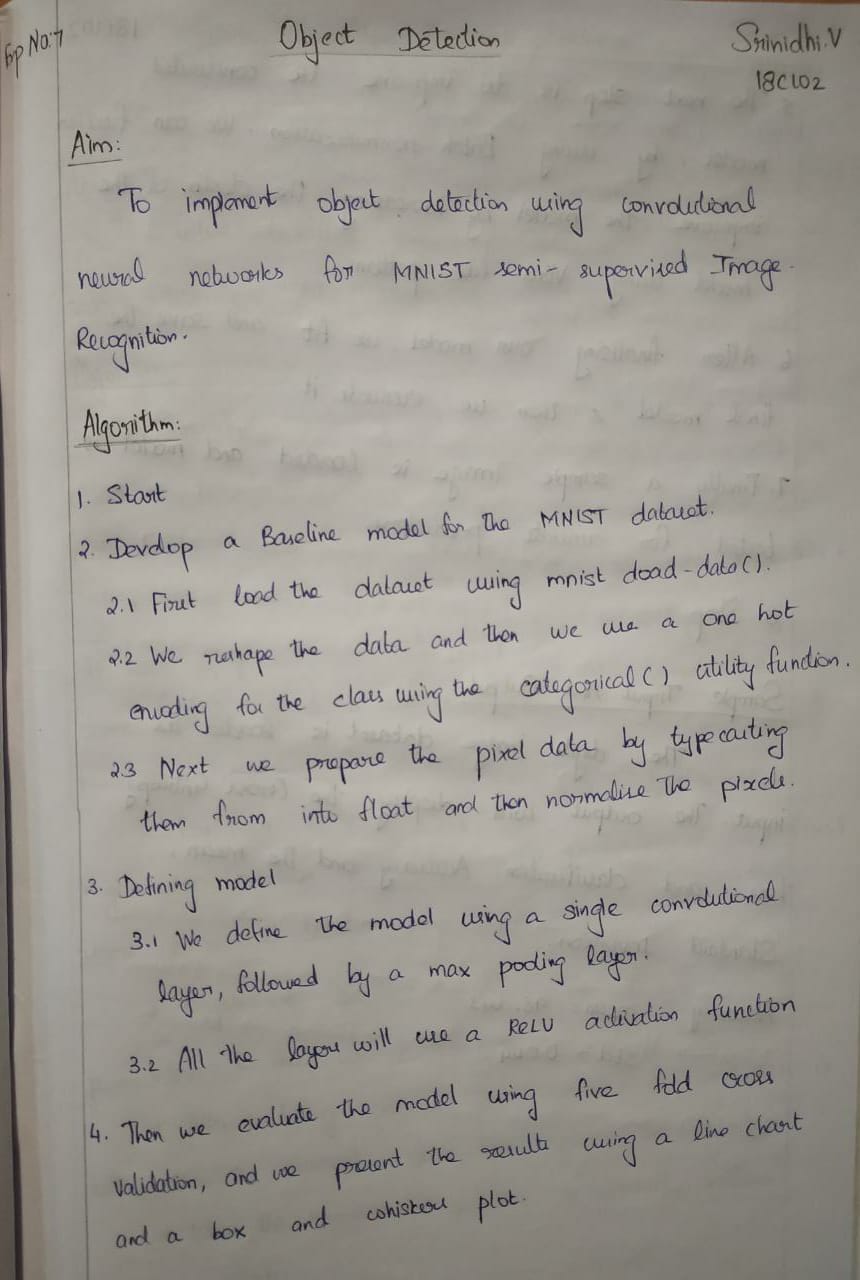
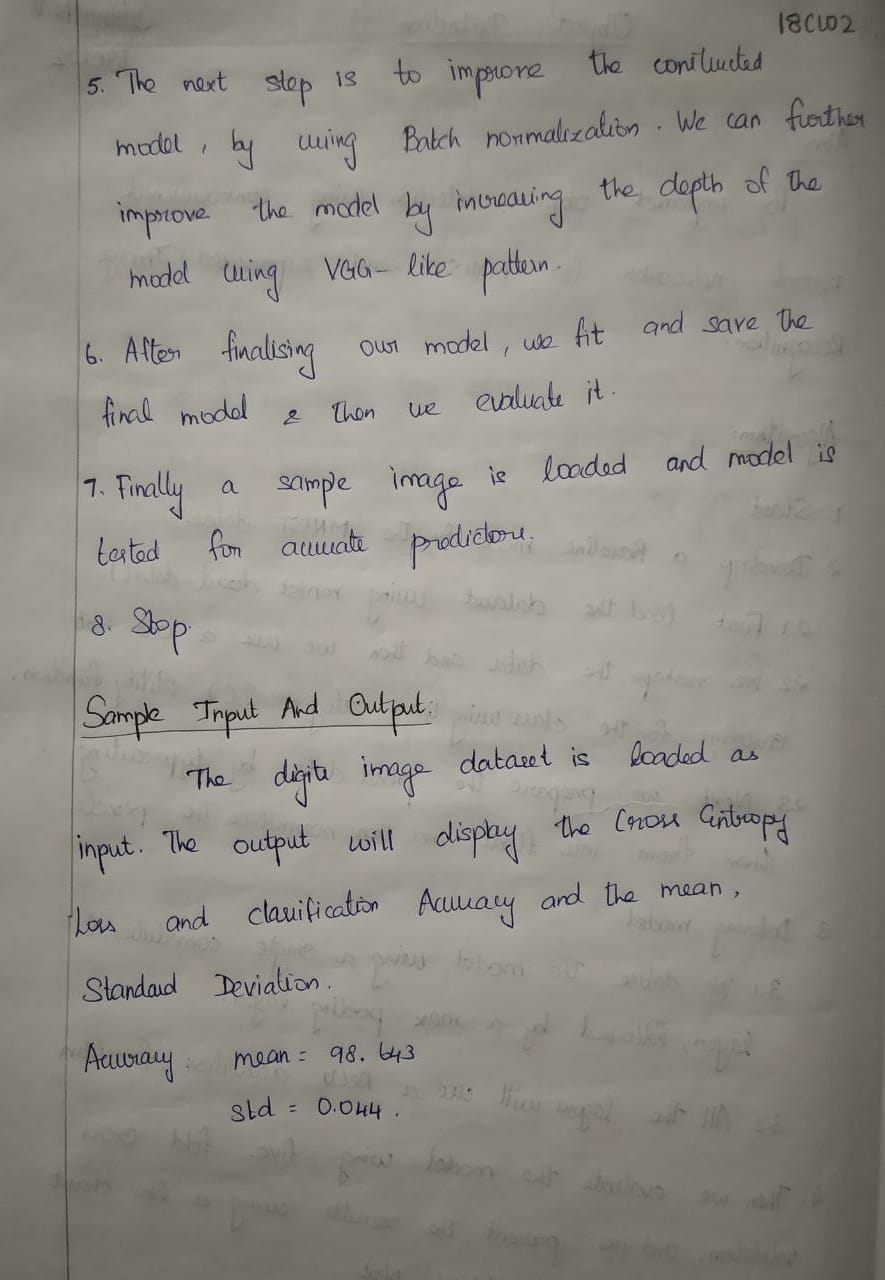
****

****

**Program:**

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

def load\_dataset():

  (trainX, trainY), (testX, testY) = mnist.load\_data()

  trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))

  testX = testX.reshape((testX.shape[0], 28, 28, 1))

  trainY = to\_categorical(trainY)

  testY = to\_categorical(testY)

  return trainX, trainY, testX, testY

def prep\_pixels(train, test):

  train\_norm = train.astype('float32')

  test\_norm = test.astype('float32')

  train\_norm = train\_norm / 255.0

  test\_norm = test\_norm / 255.0

  return train\_norm, test\_norm

def define\_model():

  model = Sequential()

  model.add(Conv2D(32, (3, 3), activation='relu', kernel\_initializer='he\_uniform', 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'))

  opt = SGD(lr=0.01, momentum=0.9)

  model.compile(optimizer=opt, loss='categorical\_crossentropy', metrics=['accuracy'])

  return model

def evaluate\_model(dataX, dataY, n\_folds=5):

  scores, histories = list(), list()

  kfold = KFold(n\_folds, shuffle=True, random\_state=1)

  for train\_ix, test\_ix in kfold.split(dataX):

    model = define\_model()

    trainX, trainY, testX, testY = dataX[train\_ix], dataY[train\_ix], dataX[test\_ix], dataY[test\_ix]

    history = model.fit(trainX, trainY, epochs=10, batch\_size=32, validation\_data=(testX, testY), verbose=0)

    \_, acc = model.evaluate(testX, testY, verbose=0)

    print('> %.3f' % (acc \* 100.0))

    scores.append(acc)

    histories.append(history)

  return scores, histories

def summarize\_diagnostics(histories):

  for i in range(len(histories)):

    pyplot.subplot(2, 1, 1)

    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='test')

    pyplot.subplot(2, 1, 2)

    pyplot.title('Classification Accuracy')

    pyplot.plot(histories[i].history['accuracy'], color='blue', label='train')

    pyplot.plot(histories[i].history['val\_accuracy'], color='orange', label='test')

  pyplot.show()

def summarize\_performance(scores):

  print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)\*100, std(scores)\*100, len(scores)))

  pyplot.boxplot(scores)

  pyplot.show()

def run\_test\_harness():

  trainX, trainY, testX, testY = load\_dataset()

  trainX, testX = prep\_pixels(trainX, testX)

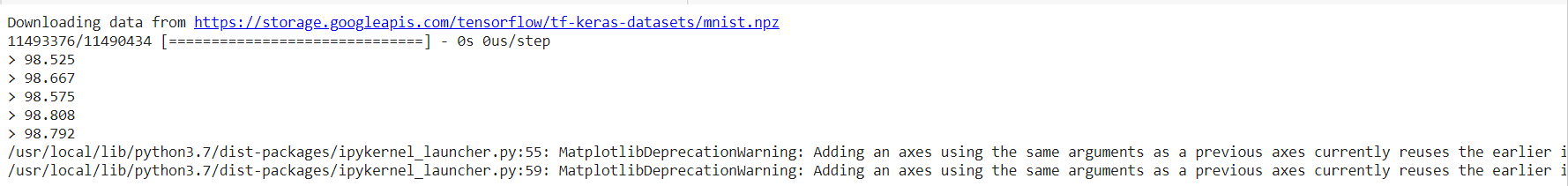
  scores, histories = evaluate\_model(trainX, trainY)

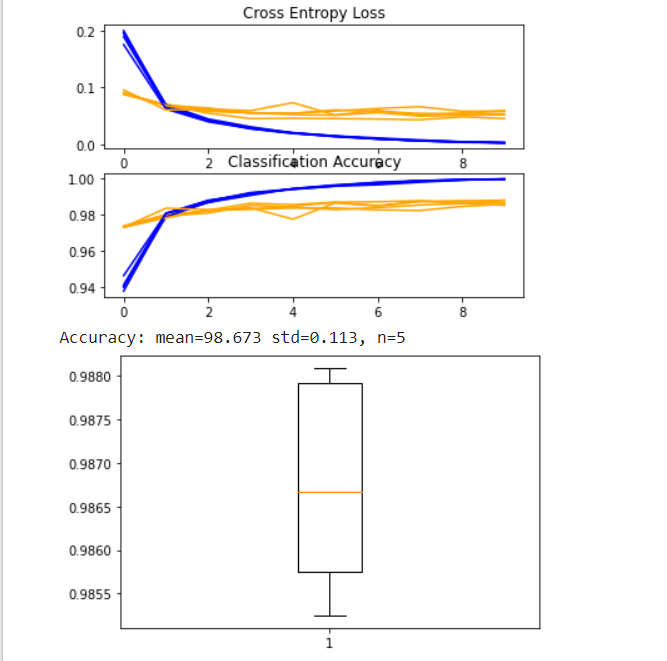
  summarize\_diagnostics(histories)

  summarize\_performance(scores)

run\_test\_harness()

**Output:**





**Result:**

Thus we have implemented object detection using convolutional networks for semi-supervised Image Recognition.