NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institute)

School of Computer Sciences & Engineering in Emerging Technologies

Deep Learning Lab (ACSML0652)

INDEX

Experiment 22

Traffic Management Using YoLo.

Code:

```
from __future__ import division                 # to allow compatibility of code between Python 2.x and 3.x with minimal overhead
from collections import Counter                 # library and method for counting hashable objects
import argparse                                 # to define arguments to the program in a user-friendly way
import os                                       # provides functions to interact with local file system
import os.path as osp                           # provides range of methods to manipulate files and directories
import pickle as pkl                            # to implement binary protocols for serializing and de-serializing object structure
import pandas as pd                             # popular data-analysis library for machine learning.
import time                                     # for time-related python functions
import sys                                      # provides access for variables used or maintained by intrepreter
import torch                                    # machine learning library for tensor and neural-network computations
from torch.autograd import Variable             # Auto Differentaion package for managing scalar based values
import cv2                                      # OpenCV Library to carry out Computer Vision tasks
import emoji
import warnings                                 # to manage warnings that are displayed during execution
warnings.filterwarnings(
    'ignore')                                   # to ignore warning messages while code execution
print('\033[1m' + '\033[91m' + "Kickstarting YOLO...\n")
from util.parser import load_classes            # navigates to load_classess function in util.parser.py
from util.model import Darknet                  # to load weights into our model for vehicle detection
from util.image_processor import preparing_image    # to pass input image into model,after resizing it into yolo format
from util.utils import non_max_suppression      # to do non-max-suppression in the detected bounding box objects i.e cars
from util.dynamic_signal_switching import switch_signal
from util.dynamic_signal_switching import avg_signal_oc_time


#*** Parsing Arguments to YOLO Model ***
def arg_parse():
    parser = argparse.ArgumentParser(
                    description=
                    'YOLO Vehicle Detection Model for Intelligent Traffic Management System')
    parser.add_argument("--images",
                    dest='images',
                    help="Image / Directory containing images to  vehicle detection upon",
                    default="vehicles-on-lanes",
                    type=str)
    parser.add_argument("--bs",
                    dest="bs",
                    help="Batch size",
                    default=1)
    parser.add_argument("--confidence_score",
                    dest="confidence",
                    help="Confidence Score to filter Vehicle Prediction",
                    default=0.3)
    parser.add_argument("--nms_thresh",
                    dest="nms_thresh",
                    help="NMS Threshhold",
                    default=0.3)
    parser.add_argument("--cfg",
                    dest='cfgfile',
```

```
                    help="Config file",
                    default="config/yolov3.cfg",
                    type=str)
    parser.add_argument("--weights",
                    dest='weightsfile',
                    help="weightsfile",
                    default="weights/yolov3.weights",
                    type=str)
    parser.add_argument(
                    "--reso",
                    dest='reso',
                    help=
                    "Input resolution of the network. Increase to increase accuracy. Decrease to increase speed",
                    default="416",
                    type=str)
    return parser.parse_args()


args = arg_parse()
images = args.images
batch_size = int(args.bs)
confidence = float(args.confidence)
nms_thesh = float(args.nms_thresh)
start = 0
CUDA = torch.cuda.is_available()

#***Loading Dataset Class File***
classes = load_classes("data/idd.names")

#***Setting up the neural network***
model = Darknet(args.cfgfile)
print('\033[0m' + "Input Data Passed Into YOLO Model..." + u'\N{check mark}')
model.load_weights(args.weightsfile)
print('\033[0m' + "YOLO Neural Network Successfully Loaded..." +
      u'\N{check mark}')
print('\033[0m')
model.hyperparams["height"] = args.reso
inp_dim = int(model.hyperparams["height"])
assert inp_dim % 32 == 0
assert inp_dim > 32
num_classes = model.num_classes
print('\033[1m' + '\033[92m' +
      "Performing Vehicle Detection with YOLO Neural Network..." + '\033[0m' +
      u'\N{check mark}')
#Putting YOLO Model into GPU:
if CUDA:
    model.cuda()
model.eval()
read_dir = time.time()

#***Vehicle Detection Phase***
try:
    imlist = [
        osp.join(osp.realpath('.'), images, img) for img in os.listdir(images)
    ]
except NotADirectoryError:
    imlist = []
    imlist.append(osp.join(osp.realpath('.'), images))
except FileNotFoundError:
    print("No Input with the name {}".format(images))
    print("Model failed to load your input.  ")
    exit()
```

```
load_batch = time.time()
loaded_ims = [cv2.imread(x) for x in imlist]

im_batches = list(
    map(preparing_image, loaded_ims, [inp_dim for x in range(len(imlist))]))
im_dim_list = [(x.shape[1], x.shape[0]) for x in loaded_ims]
im_dim_list = torch.FloatTensor(im_dim_list).repeat(1, 2)

leftover = 0

if (len(im_dim_list) % batch_size):
    leftover = 1

if batch_size != 1:
    num_batches = len(imlist) // batch_size + leftover
    im_batches = [
        torch.cat(
            (im_batches[i * batch_size:min((i + 1) *
                                           batch_size, len(im_batches))]))
        for i in range(num_batches)
    ]

write = 0

if CUDA:
    im_dim_list = im_dim_list.cuda()
start_outputs_loop = time.time()

lane_count_list = []
input_image_count = 0
denser_lane = 0
lane_with_higher_count = 0

print()
print(
    '\033[1m' +
    "----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------"
)
print('\033[1m' + "SUMMARY")
print(
    '\033[1m' +
    "----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------"
)
print('\033[1m' +
      "{:25s}: ".format("\nDetected  (" + str(len(imlist)) + " inputs)"))
print('\033[0m')
#Loading the image, if present :
for i, batch in enumerate(im_batches):
    #load the image
    vehicle_count = 0
    start = time.time()
    if CUDA:
        batch = batch.cuda()
    with torch.no_grad():
        prediction = model(Variable(batch))

    prediction = non_max_suppression(prediction,
                                     confidence,
                                     num_classes,
                                     nms_conf=nms_thesh)
```

```
    end = time.time()

    if type(prediction) == int:
        for im_num, image in enumerate(
                imlist[i * batch_size:min((i + 1) * batch_size, len(imlist))]):
            im_id = i * batch_size + im_num
            print("{0:20s} predicted in {1:6.3f} seconds".format(
                image.split("/")[-1], (end - start) / batch_size))
            print("{0:20s} {1:s}".format("Objects detected:", ""))
            print("----------------------------------------------------------")
        continue

    prediction[:,
               0] += i * batch_size  # transform the atribute from index in batch to index in imlist

    if not write:  # If we have't initialised output
        output = prediction
        write = 1
    else:
        output = torch.cat((output, prediction))

    for im_num, image in enumerate(
            imlist[i * batch_size:min((i + 1) * batch_size, len(imlist))]):
        vehicle_count = 0
        input_image_count += 1
        #denser_lane =
        im_id = i * batch_size + im_num
        objs = [classes[int(x[-1])] for x in output if int(x[0]) == im_id]
        vc = Counter(objs)
        for i in objs:
            if i == "car" or i == "motorbike" or i == "truck" or i == "bicycle" or i == "autorickshaw":
                vehicle_count += 1

        print('\033[1m' + "Lane : {} - {} : {:5s} {}".format(
            input_image_count, "Number of Vehicles detected", "",
            vehicle_count))

        if vehicle_count > 0:
            lane_count_list.append(vehicle_count)

        if vehicle_count > lane_with_higher_count:
            lane_with_higher_count = vehicle_count
            denser_lane = input_image_count

        '''print(
            '\033[0m' +
            "           File Name:     {0:20s}.".format(image.split("/")[-1]))'''
        print('\033[0m' +"           {:15} {}".format("Vehicle Type", "Count"))
        for key, value in sorted(vc.items()):
            if key == "car" or key == "motorbike" or key == "truck" or key == "bicycle":
                print('\033[0m' + "           {:15s} {}".format(key, value))

    if CUDA:
        torch.cuda.synchronize()

if vehicle_count == 0:
    print(
        '\033[1m' +
        "There are no vehicles present from the input that was passed into our YOLO Model."
    )

print(
    '\033[1m' +
```

```
    "-------------------------------------------------------------------------------------------------------------------------------------------------------
----------"
)
print(
    emoji.emojize(':vertical_traffic_light:') + '\033[1m' + '\033[94m' +
    " Lane with denser traffic is : Lane " + str(denser_lane) + '\033[30m' +
    "\n")

switching_time = avg_signal_oc_time(lane_count_list)

switch_signal(denser_lane, switching_time)

print(
    '\033[1m' +
    "-------------------------------------------------------------------------------------------------------------------------------------------------------
----------"
)
try:
    output
except NameError:
    print("No detections were made | No Objects were found from the input")
    exit()

torch.cuda.empty_cache()
```

Experiment 23

Write a program for Multivariate GRU.

Code:

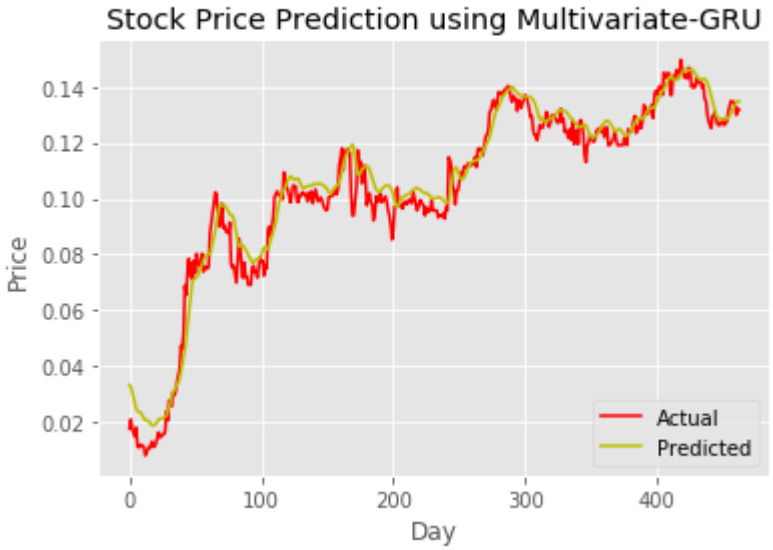
```
# Importing dependencies
import numpy as np
np.random.seed(1)
from tensorflow import set_random_seed
set_random_seed(2)
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential, load_model
from keras.layers.core import Dense
from keras.layers.recurrent import GRU
from keras import optimizers
from keras.callbacks import EarlyStopping
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt
import datetime as dt
import time
plt.style.use('ggplot')
earlystop = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=80,  verbose=1, mode='min')
callbacks_list = [earlystop]
# Loading the dataset
url = '../CSV.csv'
df = pd.read_csv(url,parse_dates = True,index_col=0)
df.tail()
#Build and train the model
def fit_model(train,val,timesteps,hl,lr,batch,epochs):
    X_train = []
    Y_train = []
    X_val = []
    Y_val = []

    # Loop for training data
    for i in range(timesteps,train.shape[0]):
        X_train.append(train[i-timesteps:i])
        Y_train.append(train[i][0])
    X_train,Y_train = np.array(X_train),np.array(Y_train)

    # Loop for val data
    for i in range(timesteps,val.shape[0]):
        X_val.append(val[i-timesteps:i])
        Y_val.append(val[i][0])
    X_val,Y_val = np.array(X_val),np.array(Y_val)

    # Adding Layers to the model
    model = Sequential()
    model.add(GRU(X_train.shape[2],input_shape = (X_train.shape[1],X_train.shape[2]),return_sequences = True,
              activation = 'relu'))
    for i in range(len(hl)-1):
        model.add(GRU(hl[i],activation = 'relu',return_sequences = True))
    model.add(GRU(hl[-1],activation = 'relu'))
    model.add(Dense(1))
    model.compile(optimizer = optimizers.Adam(lr = lr), loss = 'mean_squared_error')
```

```
    # Training the data
    history = model.fit(X_train,Y_train,epochs = epochs,batch_size = batch,validation_data = (X_val,
Y_val),verbose = 0,
                        shuffle = False, callbacks=callbacks_list)
    model.reset_states()
    return model, history.history['loss'], history.history['val_loss']

# Evaluating the model
def evaluate_model(model,test,timesteps):
    X_test = []
    Y_test = []

    # Loop for testing data
    for i in range(timesteps,test.shape[0]):
        X_test.append(test[i-timesteps:i])
        Y_test.append(test[i][0])
    X_test,Y_test = np.array(X_test),np.array(Y_test)

    # Prediction Time !!!!
    Y_hat = model.predict(X_test)
    mse = mean_squared_error(Y_test,Y_hat)
    rmse = sqrt(mse)
    r2 = r2_score(Y_test,Y_hat)
    return mse,rmse, r2, Y_test, Y_hat

# Plotting the predictions
def plot_data(Y_test,Y_hat):
    plt.plot(Y_test,c = 'r')
    plt.plot(Y_hat,c = 'y')
    plt.xlabel('Day')
    plt.ylabel('Price')
    plt.title("Stock Price Prediction using Multivariate-GRU")
    plt.legend(['Actual','Predicted'],loc = 'lower right')
    plt.show()
# Plotting the training errors
def plot_error(train_loss,val_loss):
    plt.plot(train_loss,c = 'r')
    plt.plot(val_loss,c = 'b')
    plt.ylabel('Loss')
    plt.xlabel('Epochs')
    plt.title('Loss Plot')
    plt.legend(['train','val'],loc = 'lower right')
    plt.show()
    series = df[['Close','High','Volume']] # Picking the features
print(series.shape)
print(series.tail())
# Train Val Test Split
train_start = dt.date(1997,1,1)
train_end = dt.date(2006,12,31)
train_data = series.loc[train_start:train_end]

val_start = dt.date(2007,1,1)
val_end = dt.date(2008,12,31)
val_data = series.loc[val_start:val_end]

test_start = dt.date(2009,1,1)
test_end = dt.date(2010,12,31)
test_data = series.loc[test_start:test_end]

print(train_data.shape,val_data.shape,test_data.shape)
(2515, 3) (504, 3) (503, 3)
# Normalisation
sc = MinMaxScaler()
```

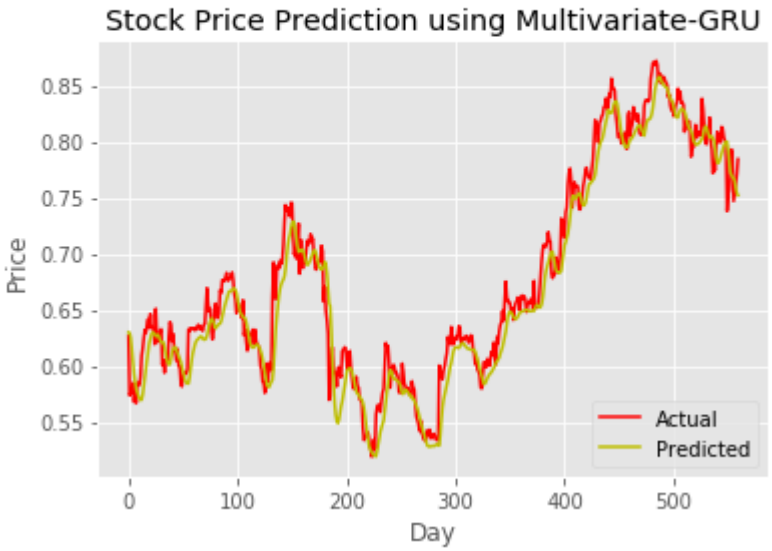
```
train = sc.fit_transform(train_data)
val = sc.transform(val_data)
test = sc.transform(test_data)
print(train.shape,val.shape,test.shape)
mse,rmse,r2_value,true,predicted = evaluate_model(model,test,40)
print("MSE =",mse)
print("RMSE =",rmse)
print("R2-Score =",r2_value)
plot_data(true,predicted)
model.save('MV3-GRU_40_[40,35]_1e-4_64.h5')
#del model  #Deletes the model
# Load a model
#model = load_model('MV3-GRU_40_[40,35]_1e-4_64.h5')

train = series[:7*split_size]
test = series[7*split_size:8*split_size]
X_train,Y_train = [],[]
# Loop for training data
for i in range(timesteps,train.shape[0]):
    X_train.append(train[i-timesteps:i])
    Y_train.append(train[i][0])
X_train,Y_train = np.array(X_train),np.array(Y_train)

start = time.time()
history = model.fit(X_train,Y_train,epochs = num_epochs,batch_size = batch_size,validation_split = 0.2,verbose =
0,
                    shuffle = False)
end = time.time()
train_loss["Split5"] = history.history['loss']
val_loss["Split5"] = history.history['val_loss']
mse, rmse, r2_value,true,predicted = evaluate_model(model,test,timesteps)
print("Split 5")
print('MSE = {}'.format(mse))
print('RMSE = {}'.format(rmse))
print('R-Squared Score = {}'.format(r2_value))
plot_data(true,predicted)
cross_val_results.append([mse,rmse,r2_value,end-start])
model.save("MV3-GRU-Split5.h5")
```

Output:

```
MSE = 4.207981713573883e-05
RMSE = 0.006486895801208682
R2-Score = 0.9610534713455121
```



```
Split 5
MSE = 0.0004509676012200079 5
RMSE = 0.0212359977684122
R-Squared Score = 0.9486727335313272
```



Experiment 24

Write a program for Autoencoder Batchsize.

Code:

```
import tensorflow as tf
import numpy as np

def get_batch(X, size):
    a = np.random.choice(len(X), size, replace=False)
    return X[a]

class Autoencoder:
    def __init_(self, input_dim, hidden_dim, epoch=500, batch_size=10, learning_rate=0.001):
        self.epoch = epoch
        self.batch_size = batch_size
        self.learning_rate = learning_rate

        # Define input placeholder
        x = tf.placeholder(dtype=tf.float32, shape=[None, input_dim])

        # Define variables
        with tf.name_scope('encode'):
            weights = tf.Variable(tf.random_normal([input_dim, hidden_dim], dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([hidden_dim]), name='biases')
            encoded = tf.nn.sigmoid(tf.matmul(x, weights) + biases)
        with tf.name_scope('decode'):
            weights = tf.Variable(tf.random_normal([hidden_dim, input_dim], dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([input_dim]), name='biases')
            decoded = tf.matmul(encoded, weights) + biases

        self.x = x
        self.encoded = encoded
        self.decoded = decoded

        # Define cost function and training op
        self.loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x, self.decoded))))

        self.all_loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x, self.decoded)), 1))
        self.train_op = tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss)

        # Define a saver op
        self.saver = tf.train.Saver()

    def train(self, data):
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
```

```
        for i in range(self.epoch):
            for j in range(500):
                batch_data = get_batch(data, self.batch_size)
                l, _ = sess.run([self.loss, self.train_op], feed_dict={self.x: batch_data})
            if i % 50 == 0:
                print('epoch {0}: loss = {1}'.format(i, l))
                self.saver.save(sess, './model.ckpt')
        self.saver.save(sess, './model.ckpt')

def test(self, data):
    with tf.Session() as sess:
        self.saver.restore(sess, './model.ckpt')
        hidden, reconstructed = sess.run([self.encoded, self.decoded], feed_dict={self.x: data})
    print('input', data)
    print('compressed', hidden)
    print('reconstructed', reconstructed)
    return reconstructed

def get_params(self):
    with tf.Session() as sess:
        self.saver.restore(sess, './model.ckpt')
        weights, biases = sess.run([self.weights1, self.biases1])
    return weights, biases

def classify(self, data, labels):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        self.saver.restore(sess, './model.ckpt')
        hidden, reconstructed = sess.run([self.encoded, self.decoded], feed_dict={self.x: data})
        reconstructed = reconstructed[0]
        # loss = sess.run(self.all_loss, feed_dict={self.x: data})
        print('data', np.shape(data))
        print('reconstructed', np.shape(reconstructed))
        loss = np.sqrt(np.mean(np.square(data - reconstructed), axis=1))
        print('loss', np.shape(loss))
        horse_indices = np.where(labels == 7)[0]
        not_horse_indices = np.where(labels != 7)[0]
        horse_loss = np.mean(loss[horse_indices])
        not_horse_loss = np.mean(loss[not_horse_indices])
        print('horse', horse_loss)
        print('not horse', not_horse_loss)
        return hidden[7,:]

def decode(self, encoding):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        self.saver.restore(sess, './model.ckpt')
        reconstructed = sess.run(self.decoded, feed_dict={self.encoded: encoding})
    img = np.reshape(reconstructed, (32, 32))
    return img
```

```
from sklearn import datasets

hidden_dim = 1
data = datasets.load_iris().data
input_dim = len(data[0])
ae = Autoencoder(input_dim, hidden_dim)
ae.train(data)
ae.test([[8, 4, 6, 2]])
```

Output:

```
epoch 0: loss = 3.8637373447418213
epoch 50: loss = 0.25829368829727173
epoch 100: loss = 0.3230888843536377
epoch 150: loss = 0.3295430839061737
epoch 200: loss = 0.24636892974376678
epoch 250: loss = 0.22375555336475372
epoch 300: loss = 0.19688692688941956
epoch 350: loss = 0.2520211935043335
epoch 400: loss = 0.29669439792633057
epoch 450: loss = 0.2794385552406311
input [[8, 4, 6, 2]]
compressed [[ 0.72223264]]
reconstructed [[ 6.87640762  2.79334426  6.23228502  2.21386957]]
: array([[ 6.87640762,  2.79334426,  6.23228502,  2.21386957]], dtype=float32)
```

Experiment 25

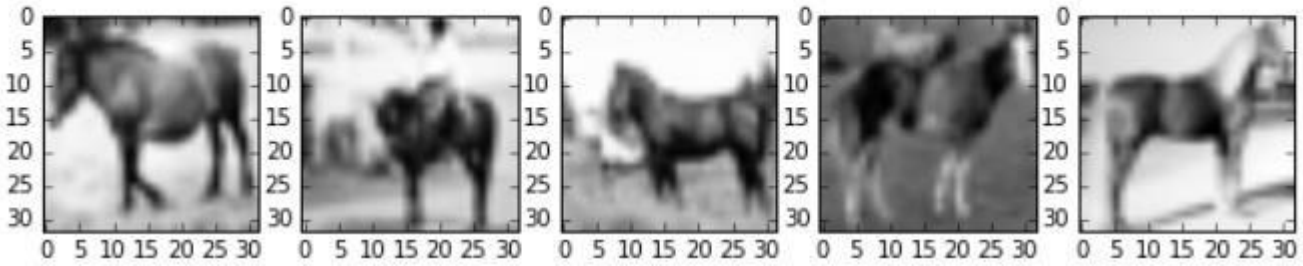
Write a program for Autoencoder with Images.

Code:

```
from matplotlib import pyplot as plt
import pickle
import numpy as np
from autoencoder import Autoencoder

def unpickle(file):
    fo = open(file, 'rb')
    dict = pickle.load(fo, encoding='latin1')
    fo.close()
    return dict

def grayscale(a):
    return a.reshape(a.shape[0], 3, 32, 32).mean(1).reshape(a.shape[0], -1)

names = unpickle('./cifar-10-batches-py/batches.meta')['label_names']
data, labels = [], []
for i in range(1, 6):
    filename = './cifar-10-batches-py/data_batch_' + str(i)
    batch_data = unpickle(filename)
    if len(data) > 0:
        data = np.vstack((data, batch_data['data']))
        labels = np.hstack((labels, batch_data['labels']))
    else:
        data = batch_data['data']
        labels = batch_data['labels']
data = grayscale(data)
x = np.matrix(data)
y = np.array(labels)
Train the autoencoder on images of horses:

horse_indices = np.where(y == 7)[0]
horse_x = x[horse_indices]
print(np.shape(horse_x))  # (5000, 3072)

print('Some examples of horse images we will feed to the autoencoder for training')
plt.rcParams['figure.figsize'] = (10, 10)
num_examples = 5
for i in range(num_examples):
    horse_img = np.reshape(horse_x[i, :], (32, 32))
    plt.subplot(1, num_examples, i+1)
    plt.imshow(horse_img, cmap='Greys_r')
plt.show()
```

Output:

```
(5000, 1024)
Some examples of horse images we will feed to the autoencoder for training
```



```
data (10000, 1024)
reconstructed (1024,)
loss (10000,)
horse 67.4191074286
not horse 65.5469002694
```