Nhóm 1:

Bài tập: Code ANN có tạo Confusion Matrix ra file csv

Link github: <https://github.com/MinhYDev1504/BaiTapDeepLearning_ANN/tree/master/ANN>

|  |  |
| --- | --- |
| #Step 01: Import các thư viện cần thiết |  |
|  | import os |
|  | import random |
|  | import numpy as np |
|  | import skimage.data |
|  | import skimage.transform |
|  | import matplotlib |
|  | import matplotlib.pyplot as plt |
|  | import tensorflow as tf |
|  | import warnings |
|  | warnings.filterwarnings("ignore") |
|  |  |
|  | #Step 02: Lấy các đường path dẫn đến tập dữ liệu Training và Testing |
|  | ROOT\_PATH = "./traffic" |
|  | train\_data\_dir = os.path.join(ROOT\_PATH, "datasets/BelgiumTS/Training") |
|  | test\_data\_dir = os.path.join(ROOT\_PATH, "datasets/BelgiumTS/Testing") |
|  | #print(os.path.abspath(train\_data\_dir)) |
|  | #---------------------------------------------- |
|  | #Step 03: Đọc/Load dữ liệu ra các mảng |
|  | # Function Load a data set and returns two lists images and corressponding labels |
|  | def load\_data(data\_dir): |
|  | """ |
|  | Loads a data set and returns two lists: |
|  | - images: a list of Numpy arrays, each representing an image. |
|  | - labels: a list of numbers that represent the images labels. |
|  | """ |
|  | # Get all subdirectories of data\_dir. Each represents a label. |
|  | directories = [d for d in os.listdir(data\_dir) |
|  | if os.path.isdir(os.path.join(data\_dir, d))] |
|  | # Loop through the label directories and collect the data in |
|  | # Declare Two lists: labels and images. |
|  | labels = [] |
|  | images = [] |
|  | for d in directories: |
|  | label\_dir = os.path.join(data\_dir, d) |
|  | file\_names = [os.path.join(label\_dir, f) |
|  | for f in os.listdir(label\_dir) if f.endswith(".ppm")] |
|  | # For each label, load it's images and add them to the images list. |
|  | # And add the label number (i.e. directory name) to the labels list. |
|  | for f in file\_names: |
|  | images.append(skimage.data.imread(f)) |
|  | labels.append(int(d)) |
|  | return images, labels |
|  |  |
|  | #---------------------------------------------- |
|  | #Step 04: HANDLING IMAGES OF DIFFERENT SIZES ==> RESIZE FOR IMAGES |
|  | print("Training Dataset is loading......") |
|  | images, labels = load\_data(train\_data\_dir) |
|  | print("Training Dataset is loaded!") |
|  | print("-------------------------------------------------------") |
|  | print("Before resize images to 32x32 [images]") |
|  | for image in images[:5]: |
|  | print("shape: {0}, \ |
|  | min: {1}, max: {2}"\ |
|  | .format(image.shape, image.min(), image.max())) |
|  | print("-------------------------------------------------------") |
|  | # Resize images to 32x32 |
|  | images32 = [skimage.transform.resize(image, (32, 32), mode='constant') |
|  | for image in images] |
|  |  |
|  | def display\_images\_and\_labels(images, labels): |
|  | """Display the first image of each label.""" |
|  | unique\_labels = set(labels) |
|  | plt.figure(figsize=(15, 15)) |
|  | i = 1 |
|  | for label in unique\_labels: |
|  | # Pick the first image for each label. |
|  | image = images[labels.index(label)] |
|  | plt.subplot(8, 8, i) # A grid of 8 rows x 8 columns |
|  | plt.axis('off') |
|  | plt.title("Label {0} ({1})".format(label, labels.count(label))) |
|  | i += 1 |
|  | \_ = plt.imshow(image) |
|  | plt.show() |
|  | #display\_images\_and\_labels(images32, labels) |
|  | # |
|  | print("-------------------------------------------------------") |
|  | print("After resize images to 32x32 [images32]") |
|  | for image in images32[:5]: |
|  | print("shape: {0}, \ |
|  | min: {1}, max: {2}"\ |
|  | .format(image.shape, image.min(), image.max())) |
|  | print("-------------------------------------------------------") |
|  |  |
|  | #plt.subplot(211) |
|  | #plt.imshow(images[0]) |
|  | #plt.subplot(212) |
|  | #plt.imshow(images32[0]) |
|  | #plt.show() |
|  |  |
|  | labels\_a = np.array(labels) |
|  | images\_a = np.array(images32) |
|  | print("labels: ", labels\_a.shape, "\nimages: ", images\_a.shape) |
|  | #-------------Minimum Viable Model-------------------------- |
|  |  |
|  | # Create a graph to hold the model. |
|  | graph = tf.Graph() |
|  | # Create model in the graph. |
|  | with graph.as\_default(): |
|  | # Placeholders for inputs and labels. |
|  | images\_ph = tf.placeholder(tf.float32, [None, 32, 32, 3]) |
|  | labels\_ph = tf.placeholder(tf.int32, [None]) |
|  |  |
|  | # Flatten input from: [None, height, width, channels] |
|  | # To Input Layer: [None, height \* width \* channels] == [None, 3072] |
|  | images\_flat = tf.contrib.layers.flatten(images\_ph) |
|  |  |
|  | # 1 Hidden layer: 1024 unit |
|  | xx1 = tf.contrib.layers.fully\_connected(images\_flat, 128, tf.nn.relu) |
|  | xx2 = tf.contrib.layers.fully\_connected(xx1, 128, tf.nn.relu) |
|  | # Output layer |
|  | # Generates logits of size [None, 62] |
|  | logits = tf.contrib.layers.fully\_connected(xx2, 62, tf.nn.relu) |
|  |  |
|  | # Convert logits to label indexes (int). |
|  | # Shape [None], which is a 1D vector of length == batch\_size. |
|  | predicted\_labels = tf.argmax(logits, 1) |
|  |  |
|  | # Define the loss function. |
|  | # Cross-entropy is a good choice for classification. |
|  | loss = tf.reduce\_mean(tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits=logits, labels=labels\_ph)) |
|  |  |
|  | # Create training op. |
|  | train = tf.train.AdamOptimizer(learning\_rate=0.001).minimize(loss) |
|  |  |
|  | # And, finally, an initialization op to execute before training. |
|  | init = tf.global\_variables\_initializer() |
|  |  |
|  | print("images\_flat: ", images\_flat) |
|  | #print("hidden layer: ",xx) |
|  | print("logits: ", logits) |
|  | print("loss: ", loss) |
|  | print("predicted\_labels: ", predicted\_labels) |
|  | print("Bat dau train") |
|  | #-------------TRAINING-------------------------- |
|  | # Create a session to run the graph we created. |
|  | session = tf.Session(graph=graph) |
|  |  |
|  | # First step is always to initialize all variables. |
|  | # We don't care about the return value, though. It's None. |
|  | \_ = session.run([init]) |
|  | for i in range(1001): |
|  | \_, loss\_value = session.run([train, loss], |
|  | feed\_dict= |
|  | {images\_ph: images\_a, |
|  | labels\_ph: labels\_a}) |
|  | if i % 100 == 0: |
|  | print("Loss: ", loss\_value) |
|  |  |
|  | #-------------USING THE MODEL-------------------------- |
|  | # Pick 10 random images |
|  | sample\_indexes = random.sample(range(len(images32)), 10) |
|  | sample\_images = [images32[i] for i in sample\_indexes] |
|  | sample\_labels = [labels[i] for i in sample\_indexes] |
|  |  |
|  | # Run the "predicted\_labels" op. |
|  | predicted = session.run([predicted\_labels], |
|  | feed\_dict={images\_ph: sample\_images})[0] |
|  | print(sample\_labels) |
|  | print(predicted) |
|  |  |
|  | # Display the predictions and the ground truth visually. |
|  | fig = plt.figure(figsize=(10, 10)) |
|  | for i in range(len(sample\_images)): |
|  | truth = sample\_labels[i] |
|  | prediction = predicted[i] |
|  | plt.subplot(5, 2,1+i) |
|  | plt.axis('off') |
|  | color='green' if truth == prediction else 'red' |
|  | plt.text(40, 10, |
|  | "Truth: {0}\nPrediction: {1}".format(truth, prediction), |
|  | fontsize=12, color=color) |
|  | plt.imshow(sample\_images[i]) |
|  |  |
|  | plt.show() |
|  |  |
|  | #-------------EVALUATION-------------------------- |
|  | # Load the test dataset. |
|  | test\_images, test\_labels = load\_data(test\_data\_dir) |
|  | # Transform the images, just like we did with the training set. |
|  | test\_images32 = [skimage.transform.resize(image, (32, 32), mode='constant') |
|  | for image in test\_images] |
|  | #display\_images\_and\_labels(test\_images32, test\_labels) |
|  | # Run predictions against the full test set. |
|  | predicted = session.run([predicted\_labels], |
|  | feed\_dict={images\_ph: test\_images32})[0] |
|  | # Calculate how many matches we got. |
|  | match\_count = sum([int(y == y\_) for y, y\_ in zip(test\_labels, predicted)]) |
|  | accuracy = match\_count / len(test\_labels) |
|  | print("Accuracy: {:.3f}".format(accuracy)) |
|  | # Close the session. This will destroy the trained model. |
|  | session.close() |
|  |  |
|  | confusion\_matrix = tf.contrib.metrics.confusion\_matrix(test\_labels,predicted) |
|  |  |
|  | with tf.Session() as sess: |
|  | import numpy as np |
|  | cm = sess.run(confusion\_matrix) |
|  | cm\_normalize = cm/cm.astype(np.float).sum(axis=1)[:, np.newaxis] |
|  | np.savetxt("confusion\_matrix.csv", cm, delimiter="\t",fmt="%d") |
|  | np.savetxt("normalize\_confusion\_matrix.csv", cm\_normalize, delimiter="\t",fmt="%5.3f") |