```
# Module for getting batches of preprocessed data for neural net training
# Copyright (C) 2017  Abien Fred Agarap
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Affero General Public License as published
# by the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Affero General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
# ==============================================================================

"""Module for data handling in the project"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

__version__ = "0.5.4"
__author__ = "Abien Fred Agarap"

from dataset.normalize_data import list_files
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix
```

```
import tensorflow as tf


def load_data(dataset):
    """Returns a tuple containing the features and labels
    in a dataset.

    Parameter
    ---------
    dataset : numpy.ndarray
      A NumPy array file containing the dataset to be loaded.

    Returns
    -------
    features : ndarray
        A numpy.ndarray with the features in the dataset as its elements.
    labels : ndarray
        A numpy.ndarray with the labels in the dataset as its elements.

    Examples
    --------
    >>> dataset = 'train_data.npy'
    >>> features, labels = data.load_data(dataset=dataset)
    >>> features
    array([[ 6.,  0.,  2., ...,  6.,  1.,  1.],
           [ 6.,  0.,  2., ...,  6.,  1.,  1.],
           [ 9.,  0.,  3., ...,  9.,  1.,  2.],
           ...,
           [ 7.,  3.,  7., ...,  1.,  1.,  1.],
        [ 6.,  0.,  2., ...,  6.,  1.,  1.],
        [ 8.,  0.,  2., ...,  2.,  1.,  1.]], dtype=float32)
    >>> labels
    array([ 1.,  1.,  1., ...,  0.,  1.,  1.], dtype=float32)

    """

    # load the data into memory
    data = np.load(dataset)

    # get the labels from the dataset
    labels = data[:, 17]
    labels = labels.astype(np.float32)

    # get the features from the dataset
    data = np.delete(arr=data, obj=[17], axis=1)
    data = data.astype(np.float32)

    return data, labels



def plot_confusion_matrix(phase, path, class_names):
    """Plots the confusion matrix using matplotlib.

    Parameter
    ---------
    phase : str
      String value indicating for what phase is the confusion matrix, i.e. training/validation/testing
    path : str
```

```
  Directory where the predicted and actual label NPY files reside
class_names : str
  List consisting of the class names for the labels

Returns
-------
conf : array, shape = [num_classes, num_classes]
  Confusion matrix
accuracy : float
  Predictive accuracy
"""

# list all the results files
files = list_files(path=path)

labels = np.array([])

for file in files:
    labels_batch = np.load(file)
    labels = np.append(labels, labels_batch)

    if (files.index(file) / files.__len__()) % 0.2 == 0:
        print(
            "Done appending {}% of {}".format(
                (files.index(file) / files.__len__()) * 100, files.__len__()
            )
        )

labels = np.reshape(labels, newshape=(labels.shape[0] // 4, 4))
```

```
print("Done appending NPY files.")

# get the predicted labels
predictions = labels[:, :2]

# get the actual labels
actual = labels[:, 2:]

# create a TensorFlow session
with tf.Session() as sess:

    # decode the one-hot encoded labels to single integer
    predictions = sess.run(tf.argmax(predictions, 1))
    actual = sess.run(tf.argmax(actual, 1))

# get the confusion matrix based on the actual and predicted labels
conf = confusion_matrix(y_true=actual, y_pred=predictions)

# create a confusion matrix plot
plt.imshow(conf, cmap=plt.cm.Purples, interpolation="nearest")

# set the plot title
plt.title("Confusion Matrix for {} Phase".format(phase))

# legend of intensity for the plot
plt.colorbar()

tick_marks = np.arange(len(class_names))
```

```
plt.xticks(tick_marks, class_names, rotation=45)
plt.yticks(tick_marks, class_names)

plt.tight_layout()
plt.ylabel("Actual label")
plt.xlabel("Predicted label")

# show the plot
plt.show()

# get the accuracy of the phase
accuracy = (conf[0][0] + conf[1][1]) / labels.shape[0]

# return the confusion matrix and the accuracy
return conf, accuracy
```