```
"""
EECS 445 - Introduction to Machine Learning
Fall 2022 - Project 2
Dogs Dataset
    Class wrapper for interfacing with the dataset of dog images
    Usage: python dataset.py
"""

import os
import random
import numpy as np
import pandas as pd
import torch
from matplotlib import pyplot as plt
from imageio import imread
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from utils import config
from skimage import data, io, filters


def get_train_val_test_loaders(task, batch_size, **kwargs):
    """Return DataLoaders for train, val and test splits.

    Any keyword arguments are forwarded to the DogsDataset constructor.
    """
    tr, va, te, _ = get_train_val_test_datasets(task, **kwargs)

    tr_loader = DataLoader(tr, batch_size=batch_size, shuffle=True)
    va_loader = DataLoader(va, batch_size=batch_size, shuffle=False)
    te_loader = DataLoader(te, batch_size=batch_size, shuffle=False)

    return tr_loader, va_loader, te_loader, tr.get_semantic_label


def get_challenge(task, batch_size, **kwargs):
    """Return DataLoader for challenge dataset.

    Any keyword arguments are forwarded to the DogsDataset constructor.
    """
    tr = DogsDataset("train", task, **kwargs)
    ch = DogsDataset("challenge", task, **kwargs)

    standardizer = ImageStandardizer()
    standardizer.fit(tr.X)
    tr.X = standardizer.transform(tr.X)
    ch.X = standardizer.transform(ch.X)

    tr.X = tr.X.transpose(0, 3, 1, 2)
    ch.X = ch.X.transpose(0, 3, 1, 2)

    ch_loader = DataLoader(ch, batch_size=batch_size, shuffle=False)
    return ch_loader, tr.get_semantic_label


def get_train_val_test_datasets(task="default", **kwargs):
    """Return DogsDatasets and image standardizer.

    Image standardizer should be fit to train data and applied to all splits.
    """
    tr = DogsDataset("train", task, **kwargs)
    va = DogsDataset("val", task, **kwargs)
    te = DogsDataset("test", task, **kwargs)

    # Resize
    # We don't resize images, but you may want to experiment with resizing
    # images to be smaller for the challenge portion. How might this affect
    # your training?
    # tr.X = resize(tr.X)
    # va.X = resize(va.X)
    # te.X = resize(te.X)

    # Standardize
    standardizer = ImageStandardizer()
    standardizer.fit(tr.X)
```

```
    tr.X = standardizer.transform(tr.X)
    va.X = standardizer.transform(va.X)
    te.X = standardizer.transform(te.X)

    # Transpose the dimensions from (N,H,W,C) to (N,C,H,W)
    tr.X = tr.X.transpose(0, 3, 1, 2)
    va.X = va.X.transpose(0, 3, 1, 2)
    te.X = te.X.transpose(0, 3, 1, 2)

    return tr, va, te, standardizer


def resize(X):
    """Resize the data partition X to the size specified in the config file.

    Use bicubic interpolation for resizing.

    Returns:
        the resized images as a list of numpy arrays.
    """
    image_dim = config("image_dim")
    image_size = (image_dim, image_dim)
    resized = []
    for i in range(X.shape[0]):
        xi = Image.fromarray(X[i]).resize(image_size, resample=2)
        resized.append(xi)
    resized = [np.asarray(im) for im in resized]

    return resized


class ImageStandardizer(object):
    """Standardize a batch of N images to mean 0 and variance 1.

    The standardization should be applied separately to each channel.
    The mean and standard deviation parameters are computed in `fit(X)` and
    applied using `transform(X)`.

    X can be in the format of a single numpy array or a list of numpy arrays.
    - If X is a single numpy array, it has
      shape (N, image_height, image_width, color_channel).
    - If X is a list of numpy arrays, it a list of N numpy arrays of
      shape (image_height, image_width, color_channel)
    """

    def __init__(self):
        """Initialize mean and standard deviations to None."""
        super().__init__()
        self.image_mean = None
        self.image_std = None

    def fit(self, X):
        """Calculate per-channel mean and standard deviation from dataset X."""
        # TODO: Complete this function
        X = np.array(X, dtype=np.float64)
        self.image_mean = np.mean(X, axis=(0, 1, 2))
        self.image_std = np.std(X, axis=(0, 1, 2))
        # print(self.image_mean)
        # print(self.image_std)

    def transform(self, X):
        """Return standardized dataset given dataset X."""
        # TODO: Complete this function
        X = np.array(X, dtype=np.float64)
        X_transformed = (X - self.image_mean) / self.image_std
        return X_transformed


class DogsDataset(Dataset):
    """Dataset class for dog images."""

    def __init__(self, partition, task="target", augment=False):
        """Read in the necessary data from disk.

        For parts 2, 3 and data augmentation, `task` should be "target".
```

```
        For source task of part 4, `task` should be "source".

        For data augmentation, `augment` should be True.
        """
        super().__init__()

        if partition not in ["train", "val", "test", "challenge"]:
            raise ValueError("Partition {} does not exist".format(partition))

        np.random.seed(42)
        torch.manual_seed(42)
        random.seed(42)
        self.partition = partition
        self.task = task
        self.augment = augment
        # Load in all the data we need from disk
        if task == "target" or task == "source":
            self.metadata = pd.read_csv(config("csv_file"))
        if self.augment:
            print("Augmented")
            self.metadata = pd.read_csv(config("augmented_csv_file"))
        self.X, self.y = self._load_data()

        self.semantic_labels = dict(
            zip(
                self.metadata[self.metadata.task ==
                              self.task]["numeric_label"],
                self.metadata[self.metadata.task ==
                              self.task]["semantic_label"],
            )
        )

    def __len__(self):
        """Return size of dataset."""
        return len(self.X)

    def __getitem__(self, idx):
        """Return (image, label) pair at index `idx` of dataset."""
        return torch.from_numpy(self.X[idx]).float(), torch.tensor(self.y[idx]).long()

    def _load_data(self):
        """Load a single data partition from file."""
        print("loading %s..." % self.partition)

        df = self.metadata[
            (self.metadata.task == self.task)
            & (self.metadata.partition == self.partition)
        ]

        if self.augment:
            path = config("augmented_image_path")
        else:
            path = config("image_path")

        X, y = [], []
        for i, row in df.iterrows():
            label = row["numeric_label"]
            image = io.imread(os.path.join(path, row["filename"]))
            edges = filters.sobel(image)
            # io.imshow(edges)
            # io.show()

            X.append(edges)
            y.append(row["numeric_label"])
        return np.array(X), np.array(y)

    def get_semantic_label(self, numeric_label):
        """Return the string representation of the numeric class label.

        (e.g., the numberic label 1 maps to the semantic label 'miniature_poodle').
        """
        return self.semantic_labels[numeric_label]


if __name__ == "__main__":
```

```
    np.set_printoptions(precision=3)
    tr, va, te, standardizer = get_train_val_test_datasets(
        task="target", augment=False)
    print("Train:\t", len(tr.X))
    print("Val:\t", len(va.X))
    print("Test:\t", len(te.X))
    print("Mean:", standardizer.image_mean)
    print("Std: ", standardizer.image_std)
```