```python
import torch
import torch as th
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

import os
import numpy as np
import os.path as osp
import json
from tqdm import tqdm
from PIL import Image

from utils import Config


class polyvore_dataset:
    def __init__(self):
        self.root_dir = Config['root_path']
        self.image_dir = osp.join(self.root_dir, 'images')
        self.transforms = self.get_data_transforms()

    def get_data_transforms(self):
        data_transforms = {
            'train': transforms.Compose([
                # transforms.CenterCrop(224),
                # transforms.RandomHorizontalFlip(),
                # transforms.RandomRotation(20),
                transforms.Resize(256),
                transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize( mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
            ]),
            'val': transforms.Compose([
                transforms.Resize(256),
                transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize( mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
            ]),
            'test': transforms.Compose([
                transforms.Resize(256),
                transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize( mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
            ]),
        }
        return data_transforms



    def create_dataset(self):
        # map id to category
        meta_file = open(osp.join(self.root_dir, Config['meta_file']), 'r')
        meta_json = json.load(meta_file)
        id_to_category = {}
        for k, v in tqdm(meta_json.items()):
            id_to_category[k] = v['category_id']

        # create X, y pairs
        files = os.listdir(self.image_dir)
        X = []; y = []
        for x in files:
```

```
        if x[:-4] in id_to_category:
            X.append(x)
            y.append(int(id_to_category[x[:-4]]))

    y = LabelEncoder().fit_transform(y)
    print('len of X: {}, # of categories: {}'.format(len(X), max(y) + 1))

    # split dataset 12% - Test, 20% - Val, 68% - Train
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
    X_train, X_val, y_train, y_val = train_test_split(\
        X_train, y_train, test_size=0.1) # 0.15 x 0.8 = 1.2
    return X_train, X_test, X_val, y_train, y_test, y_val, max(y) + 1


# For category classification
class polyvore_train(Dataset):
    def __init__(self, X_train, y_train, transform):
        self.X_train = X_train
        self.y_train = y_train
        self.transform = transform
        self.image_dir = osp.join(Config['root_path'], 'images')

    def __len__(self):
        return len(self.X_train)

    def __getitem__(self, item):
        file_path = osp.join(self.image_dir, self.X_train[item])
        return self.transform(Image.open(file_path)),self.y_train[item]

class polyvore_val(Dataset):
    def __init__(self, X_val, y_val, transform):
        self.X_val = X_val
        self.y_val = y_val
        self.transform = transform
        self.image_dir = osp.join(Config['root_path'], 'images')

    def __len__(self):
        return len(self.X_val)

    def __getitem__(self, item):
        file_path = osp.join(self.image_dir, self.X_val[item])
        return self.transform(Image.open(file_path)), self.y_val[item]


class polyvore_test(Dataset):
    def __init__(self, X_test, y_test, transform):
        self.X_test = X_test
        self.y_test = y_test
        self.transform = transform
        self.image_dir = osp.join(Config['root_path'], 'images')

    def __len__(self):
        return len(self.X_test)


    def __getitem__(self, item):
        file_path = osp.join(self.image_dir, self.X_test[item])
        item_id = file_path.rsplit('/', 1)[1].split(".")[0]
        return self.transform(Image.open(file_path)), self.y_test[item], item_id


def get_dataloader(debug, batch_size, num_workers):
    dataset = polyvore_dataset()
    transforms = dataset.get_data_transforms()
    X_train, X_test, X_val, y_train, y_test, y_val, classes = dataset.create_dataset()
    if debug==True:
        train_limit = 100000
        test_limit, val_limit = 10000, 10000

        train_set = polyvore_train(X_train[:train_limit], y_train[:train_limit], transform=transforms['train'])
        val_set = polyvore_val(X_val[:val_limit], y_val[:val_limit], transform=transforms['val'])
        test_set = polyvore_test(X_test[:test_limit], y_test[:test_limit], transform=transforms['test'])
        dataset_size = {'train': len(y_train[:train_limit]), 'test': len(y_test[:test_limit]), 'val': len(y_val[:val_limit])}
    else:
        train_set = polyvore_train(X_train, y_train, transforms['train'])
        val_set = polyvore_val(X_val, y_val, transforms['val'])
        test_set = polyvore_test(X_test, y_test, transforms['test'])
        dataset_size = {'train': len(y_train), 'test': len(y_test), 'val': len(y_val)}

    datasets = {'train': train_set, 'test': test_set, 'val': val_set}
    dataloaders = {x: DataLoader(datasets[x],
                            shuffle=True if x=='train' else False,
                            batch_size=batch_size,
                            num_workers=num_workers)
                            for x in ['train', 'test','val']}
    return dataloaders, classes, dataset_size
```