PyTorch Data Augmentation Script

Code

```
import os

import torch

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

from PIL import Image  # Import PIL for converting tensors back to images


# Data Augmentation for Training Dataset

# Define individual transformation functions

def random_horizontal_flip(img):

    transform = transforms.RandomHorizontalFlip(p=1.0)  # Always flip horizontally

    return transform(img), "Horizontal Flip"


def random_rotation(img):

    transform = transforms.RandomRotation(degrees=(-30, 30))  # Rotate within [-30°, 30°]

    return transform(img), "Rotation"


def random_color_jitter(img):

    transform = transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5, hue=0.1)

    return transform(img), "Color Jitter"


def random_grayscale(img):

    transform = transforms.RandomGrayscale(p=1.0)  # Convert to grayscale

    return transform(img), "Grayscale"
```

```
def random_crop(img):
    transform = transforms.RandomResizedCrop(size=(224, 224), scale=(0.5, 1.0))  # Random crop
    return transform(img), "Random Crop"


# List of transformation functions for easy iteration
transformations = [
    random_horizontal_flip,
    random_rotation,
    random_color_jitter,
    random_grayscale,
    random_crop
]


# Data augmentation for training dataset
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),                # Resize images to 224x224 pixels
    transforms.ToTensor(),                        # Convert image to tensor
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize pixel values
])


# No Data Augmentation for Validation and Test Datasets
test_val_transform = transforms.Compose([
    transforms.Resize((224, 224)),                # Resize images to 224x224 pixels
    transforms.ToTensor(),                        # Convert image to tensor
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize pixel values
])
```

```
# Load the datasets with appropriate transforms
train_dataset = datasets.ImageFolder(
    root='/Users/shalem/Documents/tumourtrace/clasification-roi/train',
    transform=train_transform
)
val_dataset = datasets.ImageFolder(
    root='/Users/shalem/Documents/tumourtrace/clasification-roi/val',
    transform=test_val_transform
)
test_dataset = datasets.ImageFolder(
    root='/Users/shalem/Documents/tumourtrace/clasification-roi/test',
    transform=test_val_transform
)

# Create DataLoaders
train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(dataset=test_dataset, batch_size=32, shuffle=False)

# Function to convert tensor to PIL image
def tensor_to_pil(tensor):
    unnormalize = transforms.Normalize(
        mean=[-0.5 / 0.5, -0.5 / 0.5, -0.5 / 0.5],
        std=[1 / 0.5, 1 / 0.5, 1 / 0.5]
    )
    tensor = unnormalize(tensor)
```

```
    tensor = tensor.clamp(0, 1)
    return transforms.ToPILImage()(tensor)


# Function to display different augmented versions of a sample image
def show_augmented_images(dataset, class_name):
    if class_name not in dataset.classes:
        raise ValueError(f"Class '{class_name}' not found. Available classes: {dataset.classes}")

    plt.figure(figsize=(15, 5))  # Create a figure with a fixed size

    class_idx = dataset.class_to_idx[class_name]

    for img, label in DataLoader(dataset, batch_size=1, shuffle=True):
        if label.item() == class_idx:
            sample_img = img[0]
            pil_img = tensor_to_pil(sample_img)
            break

    for i, transform_fn in enumerate(transformations):
        augmented_img, title = transform_fn(pil_img)

        plt.subplot(1, len(transformations), i + 1)
        plt.imshow(augmented_img)
        plt.axis('off')
        plt.title(title, fontsize=10)

    plt.suptitle(f'Different Augmentations for Class: {class_name}', fontsize=16)
```

```
    plt.tight_layout()

    plt.show()


# Display augmented images for the first class (e.g., 'benign')

show_augmented_images(train_dataset, class_name=train_dataset.classes[0])


# If there is more than one class, display augmented images for the second class (e.g., 'malignant')

if len(train_dataset.classes) > 1:

    show_augmented_images(train_dataset, class_name=train_dataset.classes[1])


# Function to count images in each class

def count_classes(dataset):

    class_counts = {class_name: 0 for class_name in dataset.classes}


    for _, labels in DataLoader(dataset, batch_size=1):

        for label in labels:

            class_name = dataset.classes[label.item()]

            class_counts[class_name] += 1


    return class_counts


train_counts = count_classes(train_dataset)

val_counts = count_classes(val_dataset)

test_counts = count_classes(test_dataset)


print(f'Train dataset counts: {train_counts}')

print(f'Validation dataset counts: {val_counts}')
```

print(f'Test dataset counts: {test_counts}')

Explanation

This script performs data augmentation on an image dataset using PyTorch and torchvision transforms. It is divided into several sections:

1. **Transformation Functions**: Functions such as `random_horizontal_flip`, `random_rotation`, `random_color_jitter`, and others are defined to perform various image augmentations. These functions return the transformed image and the name of the transformation.

2. **Dataset Loading**: The dataset is loaded using `ImageFolder` with different transformations for training (which includes augmentations) and validation/test sets (without augmentations). The `DataLoader` is used to load batches of images.

3. **Tensor to Image Conversion**: The function `tensor_to_pil` is defined to convert a tensor back to a PIL image. This is necessary because the augmentations are performed on PIL images.

4. **Displaying Augmentations**: The function `show_augmented_images` applies all the defined augmentations to a sample image from a specified class and displays them using `matplotlib`.

5. **Counting Classes**: A helper function `count_classes` is defined to count how many images are present in each class for the train, validation, and test datasets.

6. **Usage**: The script demonstrates how to apply augmentations and visualize the changes, as well as how to check the class distribution in the dataset.