Student Name: Mohammad Mohammad Beigi
Student ID: 99102189
Subject: Deep Learning

Deep Learning - Dr. E. Fatemizadeh
Assignment 3 - Question 3 -CNN and Vision

---

part 1

CustomDataset class is a custom dataset for PyTorch, designed to handle image data. It is intended for use in training a neural network, likely for a classification task with data augmentation.

Key Components

1. Imports:

```
from PIL import Image
import torchvision.transforms as v2
```

2. Class Definition:

```
class CustomDataset(Dataset):
```

Defines a class named `CustomDataset` that inherits from the `Dataset` class in PyTorch. The `Dataset` class is a base class for handling custom datasets in PyTorch.

3. Constructor (`__init__` method):

```
def __init__(self, data_dir, transform=None):
self.data_dir = data_dir
self.transform = transform
self.images = os.listdir(data_dir)
```

Initializes the dataset with the given `data_dir` (directory containing images) and an optional `transform` parameter for image transformations.

4. Data Loading (`__getitem__` method):

```
def __getitem__(self, idx):
img = Image.open(os.path.join(self.data_dir, self.images[idx]))
if self.transform: img = self.transform(img)
if img.shape[0] == 1: img = img.expand(3, -1, -1)
```

Retrieves and processes an individual data sample at the specified index.

5. Data Augmentation:

```
changed_img = img.clone()
transforms = [displacement, rotation, scaling]
# Implementation of transformation and label assignment
```

Randomly selects 1 to 3 transformations to apply (`num_transformations`) and generates a one-hot encoded `label` tensor indicating which transformations were applied.

6. Data Resizing:

```
return v2.Resize((224, 224), antialias=True)(img), v2.Resize((224, 224), antialias=True)(
    changed_img), label
```

Resizes both the original and transformed images to a fixed size of (224, 224) using antialiasing. The resized images and the label tensor are returned as a tuple.

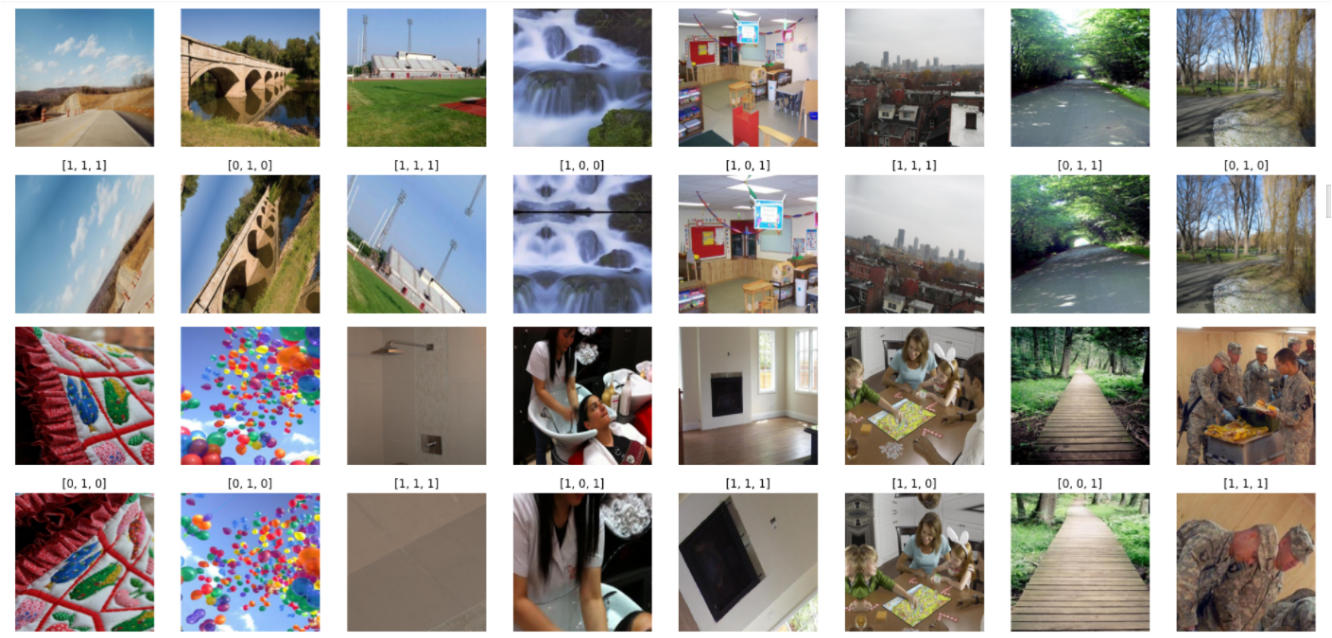
7. Dataset Length (`__len__` method):

```
def __len__(self):
return len(self.images)
```

Returns the total number of images in the dataset.
Other explanations are in comments of the code.

part 2

First and Second rows are related to train set and Third and fourth rows are related to test set.



part 3

channel out size:

```
import torch.nn as nn

class NeuralNetworkBlock(nn.Module):
def __init__(self, in_channels, out_channels):
super(NeuralNetworkBlock, self).__init__()
self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
self.conv3 = nn.Conv2d(in_channels, out_channels, kernel_size=1)
```

Forward Method

```
self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
self.conv3 = nn.Conv2d(in_channels, out_channels, kernel_size=1)
```

Here, `in_channels` and `out_channels` are the number of input channels and output channels for the convolutional layers, respectively. The `kernel_size` is set to 3 for the first two convolutional layers (`conv1` and `conv2`), and padding is set to 1 to maintain the spatial dimensions of the input.

Now, let's discuss the concept of channel out size. In convolutional neural networks (CNNs), the term "channel" typically refers to the depth or the number of filters in a convolutional layer. The channel out size is the number of output channels produced by a convolutional layer.

In your code, the output channels for the first two convolutional layers (`conv1` and `conv2`) are both set to `out_channels`. For the third convolutional layer (`conv3`), the output channels are also set to `out_channels`. The channel out size for these layers is determined by the value of `out_channels`.

In the forward method:

```
out = self.dropout(self.relu(self.batchnorm1(self.conv1(x))))
out = self.batchnorm2(self.conv2(out))
out_byPass = self.batchnorm3(self.conv3(x))
out = self.dropout(self.relu(out + out_byPass))
```

The input `x` undergoes convolution, batch normalization, dropout, and ReLU activation in the first two lines. Additionally, the original input `x` undergoes convolution and batch normalization in the third line. The final output `out` is a combination of the processed output and the bypassed output.

In summary, the channel out size is determined by the `out_channels` parameter, and it remains constant across the convolutional layers within the block. The output tensor has a depth or number of channels equal to the specified `out_channels`.

padding = 1:

Avoiding the loss of border information because of skip connections: Without padding, the pixels at the border of the input image may be underrepresented in the convolution operation, as they are not covered by the kernel entirely. Padding helps in including these border pixels in the convolution process.

conv 1 by 1:

1. **Dimensionality Reduction:**
    - 1x1 convolutions can be used to reduce the number of channels (depth) in the input tensor. This helps in reducing the computational cost and the number of parameters in the network. It's particularly useful in scenarios where a large number of channels are not necessary.
2. **Non-linearity Introducing:**

- Even though the convolution is 1x1, it still involves non-linear activation functions (like ReLU or sigmoid). This introduces non-linearity to the model, which can be crucial for learning complex relationships in the data.

In summary, 1x1 convolutions are a versatile tool in neural network architectures. They provide a way to control the depth, introduce non-linearity, and adjust the computational complexity of the model, all while maintaining efficiency. They are particularly useful in designing deep and efficient neural networks.

part 4

In summary, images with 2 changes are more difficult to detect than images with 1 change. Also images with 3 changes are the most difficult one to detect.