Documentation for Image Data Augmentation

TensorFlow

Overview

To achieve data augmentation on images using TensorFlow, we used the `ImageDataGenerator`. It applies several transformations to generate new images from existing ones, enhancing the dataset for training machine learning models. The transformations include rotation, width shift, height shift, zoom, and horizontal flip. The augmented images are saved to a specified directory.

Requirements

- TensorFlow
- NumPy
- PIL (Python Imaging Library)
- os module

Steps

1. Import Necessary Libraries

First, import the required libraries.

2. Create Output Directory

Create a directory to save the augmented images.

3. Load and Preprocess Images

Load images from a specified folder and convert them to NumPy arrays.

4. Define ImageDataGenerator with Transformations

Set up the `ImageDataGenerator` with specified transformations.

5. Generate Augmented Images

Generate and save augmented images for each input image. In this script, five augmented images are generated per input image. At first the images augmented on are all gotten from memory. This is not very efficient as with large datasets, its impossible to load all the images in memory.

6. Process Each Image Individually (for Efficient Handling)

To handle images efficiently, especially with larger datasets, process each image one at a time, therefore only loading 1 at a time in memory

7. Process Images in Batches(also for Efficient Handling)

Another way to handle images efficiently is to load them in smaller batches, therefore having less images in memory, but still faster than 1 at a time.

Explanation of Transformations

- **rotation_range=40**: Randomly rotates images within the range of -40 to +40 degrees.
- **width_shift_range=0.2**: Shifts images horizontally by up to 20% of the width.
- **height_shift_range=0.2**: Shifts images vertically by up to 20% of the height.
- **zoom_range=0.2**: Zooms in on images by up to 20%.
- **horizontal_flip=True**: Randomly flips images horizontally.
- **fill_mode='nearest'**: Fills newly created pixels after transformations with the nearest pixel values from the boundary of the original image.

Conclusion

This script demonstrates how to efficiently augment image data using TensorFlow's `ImageDataGenerator`. It loads images from a directory, applies specified transformations, and saves multiple augmented versions of each image to a new directory. The process is designed to handle large datasets efficiently by processing one image at a time.

Use of Generative AI

Generative AI was used to generate this documentation. The code was copy and pasted into chat-gpt for which a layout to use was generated. Then the document was read through and was modified to ensure it was accurate to the code. Some sections were also added.

Q3: Data Augmentation using PyTorch

Steps and Implementation

1. **Importing Libraries:** We started by importing necessary libraries, including os for directory operations, PIL for image processing, torchvision.transforms for applying transformations, and metrics from skimage for image comparison.
2. **Reading Images:** We loaded the images from the specified directory and stored them in a list.
3. **Applying Transformations:** We defined a set of transformations using transforms.Compose(). These transformations included random affine transformations, horizontal flips, rotations, and color jittering.
4. **Generating Augmented Images:** For each image, we generated five augmented versions using the defined transformations. These images were saved in a new directory.
5. **Comparing Images:** We compared the original and augmented images using SSIM and MSE metrics. The results were stored in a DataFrame for analysis.

Results and Analysis

- **SSIM Results:** The SSIM values were consistently below 0.5, indicating significant differences between the original and augmented images.
- **MSE Results:** The MSE values were high, further confirming the substantial differences.

These results suggest that the transformations introduced significant variations in the images, effectively augmenting the dataset.

Q4: Photometric Distortions using OpenCV

Steps and Implementation

1. **Importing Libraries:** Similar to Q3, we imported necessary libraries for image processing and comparison.
2. **Defining Photometric Functions:** We implemented functions to adjust brightness, contrast, and saturation, as well as shift color channels.
3. **Applying Transformations:** We applied these functions to images and saved the results.
4. **Comparing Images:** We compared the original and augmented images using SSIM and MSE metrics, storing the results in a DataFrame.
5. **Visualizing Results:** We used matplotlib to visualize the original and augmented images for each photometric adjustment.

Results and Analysis

- **SSIM Results:** The SSIM values were close to 1 for most transformations, indicating high similarity with the original images.
- **MSE Results:** The MSE values were low, further confirming the high similarity.

These results suggest that the implemented photometric transformations were effective in maintaining the overall appearance of the images while introducing necessary variations.

Use of Generative AI

Generative AI was used to enhance this documentation based on the provided notes. Specifically, ChatGPT assisted in:

- Refining the explanation of photometric distortion techniques.
- Suggesting effective methods for comparing augmented images.
- Providing guidance on structuring the report.

The use of generative AI has been declared in the Generative AI Journal and followed the project's ethical guidelines.