Q3: Data Augmentation in mainstream machine learning frameworks Documentation

Data augmentation in TensorFlow and PyTorch

Composed by:

Mark Dingli (ID: 20703H)

Pedro A. H. Guidobono (Student ID: 2100273)

Table of Contents

1. TensorFlow

Check where we are stepping into. Search for information on ImageDataGenerator and how to use it in order to perform data augmentation techniques.
Firstly, a visit to the official TensorFlow website documentation was in place to check for useful input and guidance on how to use it.
On the website, we search for ImageDataGenerator, where we end up in the [1].
On that page, we see a message stressing that **ImageDataGenerator is Deprecated** and it is not recommended for new codes. It is preferable to load images with *tf.keras.utils.image_dataset_from_directory* and transform the output with preprocessing layers, it contains a link with tutorials.

For the sake of this assignment, we continue with the ImageDataGenerator library.
On that same page, we can find the link [2] which contains steps taken in notebooks. However, there is too much unnecessary information resulting in a waste of time to filter and obtain the relevant guidance for this assignment. Therefore, we searched for more straightforward guidance and thanks to [3], we could have a better understanding of the implementation.
With the simplicity of TensorFlow, there was not much more we needed to research in order to complete the task for this assignment.

To simplify, there are 3 main steps to perform data augmentation with ImageDataGenerator:

- **Import libraries.**
- Set wanted transformations with **ImageDataGenerator()**
- Generate and save batches of augmented images using the function **.flow()**

1.1 Imported libraries

The main libraries to use ImageDataGenerator are the following:

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
```

Figure 1 - Libraries to work with ImageDataGenerator.

Note: Tensorflow must first be installed

We can check if a package is installed from the terminal with “pip show”.

```
(compVisionPartB) C:\Users\pedro>pip show tensorflow
Name: tensorflow
Version: 2.12.0
```

Figure 2 - Check version of TensorFlow if installed.

In case the package is now found, use pip to install it.

```
(compVisionPartB) C:\Users\pedro>pip install tensorflow
```

Figure 3 - Command to install TensorFlow.

Libraries used to build the code structure:

```
from skimage import io  # pip install scikit-image
                        # Used to load image into a variable
import numpy as np
import os               # Help retrieve directory infomation
from PIL import Image   # Contain tools to work with images
```

Figure 4 - Other libraries that are used for the implementation.

Libraries used for comparison:

```
import cv2
from skimage.metrics import structural_similarity as strucSim
from matplotlib import pyplot as plt
```

Figure 5 - Libraries that are used to compare the original image with augmented images.

1.2 ImageDataGenerator()

Below we have a table with the parameters we used for the transformations settled in the ImageDataGenerator. The full table can be found in [1]. We created the variable **data_generator** to store the ImageDataGenerator() with these parameters.

Parameter	Description
rotation_range	Int. Degree range for random rotations.
width_shift_range	Float, 1-D array-like or int float: fraction of total width, if < 1, or pixels if >= 1. 1-D array-like: random elements from the array. int: integer number of pixels from interval (-width_shift_range, +width_shift_range) - With width_shift_range=2 possible values are integers [-1, 0, +1], same as with width_shift_range=[-1, 0, +1], while with width_shift_range=1.0 possible values are floats in the interval [-1.0, +1.0).
height_shift_range	Float, 1-D array-like or int float: fraction of total height, if < 1, or pixels if >= 1. 1-D array-like: random elements from the array. int: integer number of pixels from interval (-height_shift_range, +height_shift_range) - With height_shift_range=2 possible values are integers [-1, 0, +1], same as with height_shift_range=[-1, 0, +1], while with height_shift_range=1.0 possible values are floats in the interval [-1.0, +1.0).
brightness_range	Tuple or list of two floats. Range for picking a brightness shift value from.
shear_range	Float. Shear Intensity (Shear angle in counter-clockwise direction in degrees)
zoom_range	Float or [lower, upper]. Range for random zoom. If a float, [lower, upper] = [1-zoom_range, 1+zoom_range].
fill_mode	One of {"constant", "nearest", "reflect" or "wrap"}. Default is 'nearest'. Points outside the boundaries of the input are filled according to the given mode: * **'constant'**: kkkkkkkk | abcd | kkkkkkkk (cval=k) * **'nearest'**: aaaaaaaa | abcd |dddddddd * **'reflect'**: abcddcba| abcd |dcbaabcd * **'wrap'**: abcdabcd| abcd |abcdabcd
horizontal_flip	Boolean. Randomly flip inputs horizontally.
rescale	rescaling factor. Defaults to None. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (after applying all other transformations).

Table 1 - Description of parameters accepted by ImageDataGenerator(). [1]

ImageDataGenerator also provides the optional parameter "**fill_mode".** When an image is shifted or translated, Some parts of the resulting image are not filled in. The **fill_mode** parameter helps us decide what to do with those parts. We have four options, *"constant", "nearest", "wrap",* and the one we decided to use*, "reflect".* We decided to use the reflected option since it is the one that always resulted in images with at least part of the main object in the image.

1.3 Makedir

A loop was run to retrieve the names of each directory where data will be collected and make new directories with the same names to output the augmented images classifying by keeping them separated with the respective folder's name.

1.4 .flow()

There are three options to use **flow()** in order to use the specs from ImageDataGenerator to generate and save new transformed images:

.flow_from_directory()

.flow()

.flow_from_dataframe()

First, we tried the function "**data_generator.flow_from_directory()"** (remember that **data_generator** is the variable we created to store ImageDataGenerator()). Since flow_from_directory() is simple to loop over all subdirectories of a parent dir and perform the image augmentation and the code also ends up looking cleaner. However, even with the parameter "class_mode" which can help us save the images with names that can help with the categorization, we failed to save images with the original name. The options for this parameter save the images with a different type of identifier, for example, "numbers".

We then used "**data_generator.flow()"**. With a strategy in a loop, we managed to save each batch of images within the correct subfolder name, also naming each new image with the correct object name and some auxiliary info on the name to identify that it originated from the data augmentation process.

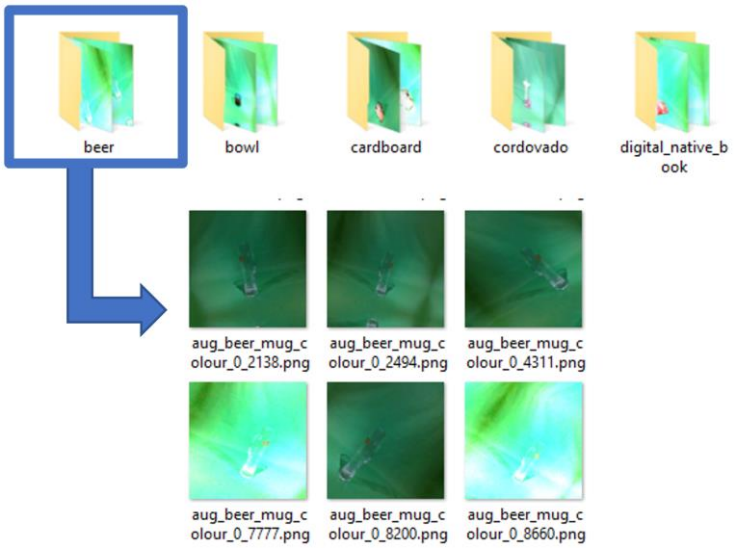

Figure 6 - Inside folder "./Augmented/beer/"

ImageDataGenerator is known for helping on saving memory during image processing, however, we failed to find a trusted font to give information about which of those three methods would be the best for saving memory. On the other hand, when asking ChatGPT, it affirms that **flow_from_directory()** needs to load all the images within a folder into memory. That would be fine for this project, however, if it is a big dataset, or images are of a huge size, that could be a problem depending on the machine.
In contrast "**.flow()"** loads one image at a time, which makes it clear to be an optimal option regarding memory.

The third method, "**.flow_from_dataframe()".** Images may be stored together in a single folder, accompanied by a CSV or JSON file that associates the image filenames with their respective classes. With this method, you can conveniently augment images by extracting their name and target value directly from a data frame [4].

2. PyTorch

2.1 Imported libraries

```
import torch
import torchvision.transforms as transforms
from PIL import Image
import os
import random
import logging
import matplotlib.pyplot as plt
import numpy as np
```

Figure 7 - PyTorch Imported Libraries

- **torch** is the main library for tensor computations.
- **torchvision.transforms** provides various image transformations
- **PIL** from the Pillow library is used for image processing
- **os** is used for file and directory operations.
- **random** is used for random number generation.
- **logging** is used for logging information and errors.
- **matplotlib.pyplot** is used for plotting and visualization.
- **numpy** provides array operations and numerical computations.

2.2 Image Transformations

The ***torchvision.transforms*** module provides several built-in transformation functions for image augmentation in PyTorch [5]. Below we discuss the transformations used in the updated code:

- **RandomHorizontalFlip:** This function randomly flips the image horizontally with a probability of 0.5. It helps to increase the dataset's diversity and improve the model's ability to generalize.

- **RandomAffine:** This function applies a random affine transformation to the image, which includes rotation, translation, scaling, and shearing. The range of rotation is from -20 to 20 degrees. The translate parameter is a tuple of maximum absolute fraction for horizontal and vertical translations, and we have set it to (0.1, 0.1). The scaling is between 0.8 to 1.2, and shearing is between -10 to 10 degrees. Affine transformations help the model to be more robust to changes in the object's position, scale, and orientation in the image.

- **ColorJitter:** This function randomly changes the brightness, contrast, and saturation of an image. We have set the amount of change to 0.2 for all three attributes. Color variations can help the model to recognize the objects under different lighting conditions.

- **RandomPerspective:** This function applies a random perspective transformation to the image. The distortion scale is set to 0.2 and the probability of applying the transformation is 0.5. Perspective transformations can help the model to recognize the objects from different angles of view.

The transformation pipeline was defined by composing a list of the above-mentioned transformation functions using **transforms.Compose()**. The images were passed through this pipeline to obtain augmented images.

2.3 Image Augmentation

After setting up the transformation pipeline, the image augmentation process can be started [6]. The path to the directory containing the original dataset images was set using the *image_directory* variable. The output directory, where the augmented images would be saved, was set using the *output_directory* variable. If the output directory did not exist, it was created. Each image from the COTS dataset was subjected to the transformation pipeline to generate augmented versions of the original image.

The *augment_images* function was defined to perform the data augmentation. It iterated over the images in the input directory, applied the specified transformations using the *transforms.ToTensor* and *transforms.ToPILImage* functions, and saved the augmented images to the output directory using the save method of the PIL Image object. The *augment_images* function was called, passing the input directory, output directory, transformation sequence, and the number of augmentations as arguments. The function processed each image, applied the transformations, and saved the augmented images with appropriate filenames. The augmentation was done five times for each image, resulting in five augmented images per original image.

To ensure the reproducibility of the augmentation process, a random seed value was set. The seed value is used to generate a sequence of random numbers that are used in the transformation functions. By using the same seed value, the same sequence of random numbers is generated every time the code is run, ensuring that the augmentation process is consistent and reproducible.

2.4 Folders & Files

The original images used for augmentation are stored in a folder named *"COTS_Dataset"*. This folder contains the selected base images from the COTS dataset, each of which has been subjected to the transformation pipeline for augmentation. The ten original images are the following:

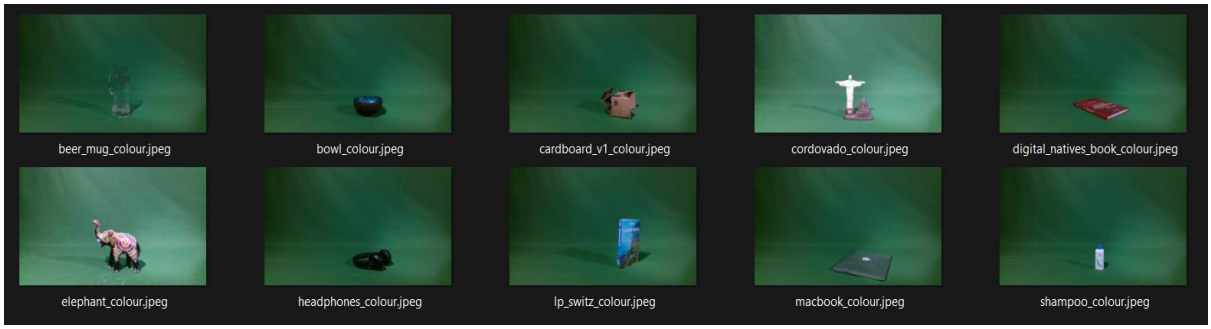

Figure 8 - Original Images from COTS Dataset

The augmented images resulting from the transformation process are saved in a separate directory titled *"Augmented_images"*. In this folder, each augmented image is saved with a filename indicating the transformation applied and the original image's name. This organization allows for easy identification of the relationship between original and augmented images.

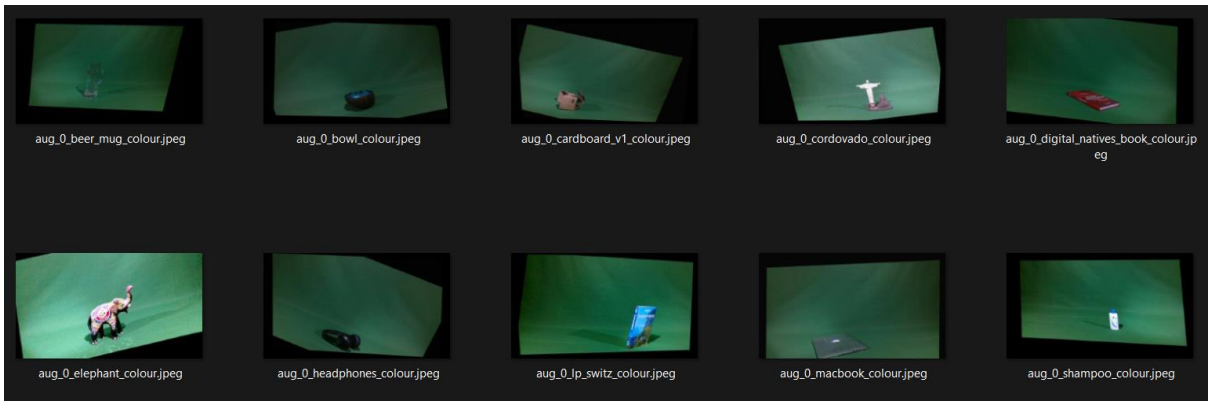

Figure 9 - Snippet of the first 10 Augmented images

3. Comparison

After the augmented images were generated, it was necessary to evaluate how different the augmented images are from the original images. This was done by employing comparison metrics, which quantitatively measure the difference between two images.

- **Pixel Difference:** This is a simple metric that calculates the absolute difference in pixel values between two images.

- **Structural Similarity Index (SSIM):** This is a more sophisticated metric that considers changes in texture, brightness, and contrast to evaluate the similarity between two images.

These metrics provide an understanding of how the transformations have altered the images from their original state.

For larger datasets, we would typically utilize the FLANN-based Matcher to compare images. However, given the smaller size of the current image dataset, we chose to implement Brute-Force Matching with SIFT Descriptors and Ratio Test, as discussed in our fourth tutorial.

Additionally, we employed the *structural_similarity* function from the *skimage.metrics* library to assess the similarity of each augmented image to its original counterpart. This library, initially suggested by ChatGPT [7], provided the foundational code that we further refined to resize images and print similarity results in percentage format.

The Structural Similarity Index (SSIM) was employed to quantitatively compare the augmented images to their originals. This holistic metric assesses changes in structural information, luminance, and contrast, providing a comprehensive understanding of the degree of transformation applied. SSIM values range from -1 to 1, with 1 indicating identical images. Lower SSIM scores in our results signify that significant, yet meaningful transformations were applied, resulting in substantially different but recognizable versions of the original images.

4. References

[1] TensorFlow.org, "tf.keras.preprocessing.image.ImageDataGenerator," 2023. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator (accessed May 28, 2023).

[2] TensorFlow.ork, "tf.data: Build TensorFlow input pipelines," 2023. https://www.tensorflow.org/guide/data (accessed May 28, 2023).

[3] S. Bhattiprolu, "127 - Data augmentation using keras," 2020. https://www.youtube.com/watch?v=ccdssX4rIh8&ab_channel=DigitalSreeni (accessed May 28, 2023).

[4] A. Bhandari, "Image Augmentation on the fly using Keras ImageDataGenerator!," 2020. https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/ (accessed May 28, 2023).

[5] "Transforming and augmenting images," Transforming and augmenting images - Torchvision 0.15 documentation, https://pytorch.org/vision/stable/transforms.html (accessed May 29, 2023).

[6] A. Rosebrock, "Keras ImageDataGenerator and data augmentation," PyImageSearch, https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/ (accessed May 29, 2023).

[7] "ChatGPT," Introducing ChatGPT, https://openai.com/blog/chatgpt (accessed May 29, 2023).