Building a data augmentation pipeline for training efficient deep learning architectures in medical imaging segmentation: A study using 4D MRI volumes from the Decathlon dataset

Poster No.: C-11960

Congress: ECR25

Authorblock: D. A. Motoc; Oradea/RO

Disclosures:
Dan Adrian Motoc: Nothing to disclose

Keywords: Artificial Intelligence, Neuroradiology brain, Oncology, Image manipulation / Reconstruction, MR, Neural networks, Computer Applications-Detection, diagnosis, eLearning, Segmentation, Cancer, Neoplasia

Purpose or Learning Objective:

The aim of this study was to develop an efficient data augmentation pipeline to address the challenges of small medical imaging datasets, specifically for brain tumour segmentation. By focusing on coding efficiency and optimization, the objective was to create a scalable process capable of generating diverse training samples, enhancing future model development and diagnostic performance.

In the field of medical imaging, one of the significant challenges is the limited availability of annotated datasets, especially for specific conditions such as brain tumors. This scarcity of data hampers the development and training of robust deep learning models, which require large and diverse datasets to generalize well. The purpose of our study is to design and implement a data augmentation pipeline that can generate a wide variety of training samples from a limited dataset, thereby addressing this challenge.

Our specific objectives include:

- developing a set of augmentation techniques tailored for 4D MRI volumes, including both 3D volumetric and 2D slice-based augmentations.

- ensuring that the augmented images maintain clinical relevance and anatomical accuracy.

- creating a scalable and efficient pipeline that can be easily integrated into existing deep learning workflows.

- evaluating the effectiveness of the augmentation techniques in enhancing the diversity of the training data and potentially improving model performance in future studies.

Methods or Background:

This retrospective study used 500 training images and 500 corresponding segmentation masks from the Decathlon brain tumour dataset. The dataset consisted of 4D MRI volumes of patients with confirmed brain tumours. The pipeline was developed in Python, applying both 3D and 2D augmentations. NIfTI images were loaded with NiBabel, converted into PyTorch tensors, and processed. The 3D augmentations included affine transformations, intensity rescaling, and random flipping, using TorchIO. For 2D data, slices were extracted and further augmented with rotations, cropping, and intensity normalisation. A custom visualisation function (plot_tensor) and NiLearn were used to verify anatomical accuracy.

I. Dataset

The dataset utilized in this study is the Decathlon brain tumor dataset, a well-known benchmark dataset in the medical imaging community. It consists of 4D MRI volumes, where the 1st dimension represents different imaging sequences (such as T1, T2, FLAIR, and Diffusion), while the other three dimensions correspond to the 3D spatial coordinates (X, Y, Z) that define the anatomical structure of the brain. This 4D matrix captures both the spatial information and the variations across different imaging modalities. The dataset includes a total of 500 training images and their corresponding segmentation masks, which provide detailed annotations of the tumor regions. These images represent various stages and types of brain tumors, including gliomas, meningiomas, and metastases, thereby offering a comprehensive set of data for the development and validation of our data augmentation pipeline.

II. Pipeline Development

The data augmentation pipeline was meticulously developed using Python, a versatile and widely-used programming language in the field of medical imaging and deep learning. Several specialized libraries were leveraged to ensure efficient data processing and augmentation.

1. Loading and Preprocessing:

- NiBabel: we employed NiBabel, a robust library designed for accessing and manipulating neuroimaging data formats. NiBabel was used to load the NIfTI (Neuroimaging Informatics Technology Initiative) images into our pipeline, as the samples from the Decathlon dataset were provided in this format. These images were then preprocessed to ensure they were in a consistent format suitable for augmentation.

- conversion to PyTorch Tensors: the loaded NIfTI images were converted into PyTorch tensors to facilitate efficient processing and manipulation within the PyTorch framework. This conversion was crucial for leveraging PyTorch's powerful capabilities for deep learning and tensor operations.

2. Augmentation Techniques

2.1. 3D Augmentations

- Affine Transformations: we applied random affine transformations, including translations, rotations, and scaling, to introduce geometric variability in the images. These transformations help simulate different orientations and positions of the brain, thereby enhancing the robustness of the training data.

- Intensity Rescaling: this technique involved adjusting the intensity values of the images to simulate different imaging conditions and enhance contrast. By varying the intensity, we aimed to mimic variations in MRI acquisition parameters and patient conditions.

- Random Flipping: random flipping of the images along different axes was performed to increase the diversity of the training samples. This simple yet effective augmentation technique helps the model learn to recognize tumors from various perspectives.

- Implementation with TorchIO: these 3D augmentations were implemented using TorchIO, a specialized library for medical image augmentation. TorchIO provides a comprehensive set of tools for applying complex transformations to 3D medical images, ensuring high efficiency and ease of use.

2.2. 2D Augmentations

- Slice Extraction: to further enhance the diversity of the dataset, we extracted 2D slices from the 3D volumes. This approach allows us to apply additional augmentations to individual slices, thereby increasing the number of training samples.

- Rotations and Cropping: random rotations and cropping were applied to the 2D slices to introduce spatial variability. These augmentations help the model become invariant to minor changes in the orientation and position of the tumor.

- Intensity Normalization: this involved normalizing the intensity values of the slices to a standard range, ensuring consistency across the dataset. Intensity normalization is essential for mitigating variations due to different imaging conditions and scanners.

- Application to Individual Slices: by applying these augmentations to individual slices, we significantly enhanced the diversity of the training data, making the dataset more representative of real-world scenarios.

3. Validation:

- Custom Visualization Function (plot_tensor): to ensure the anatomical accuracy and clinical relevance of the augmented images, we developed a custom visualization function named plot_tensor. This function allows us to visually inspect the augmented images and compare them with the original images. By doing so, we can verify that the augmentations do not distort the critical anatomical structures, preserve the integrity necessary for medical diagnosis, and ensure that the segmentation mask accurately matches the spatial position of the original tumor site.

- Verification with NiLearn: NiLearn, a neuroimaging library, was utilized to further validate the augmented images. NiLearn provides tools for advanced analysis and visualization of neuroimaging data, ensuring that the augmentations preserve the clinical relevance of the images. This step is crucial for confirming that the augmented images are suitable for training deep learning models for medical imaging segmentation.

Results or Findings:

The pipeline generated diverse variations of 4D MRI volumes by acquiring and augmenting 3D, 2D slices. A key focus was the alignment between augmented images and segmentation masks, which remained accurate throughout transformations. Visualisation confirmed that the dataset maintained clinical relevance while enhancing variability, crucial for deep learning model generalisation. Diagnostic accuracy metrics, such as specificity or sensitivity, were not calculated but will be evaluated in future model comparisons.

1. Data Augmentation:

The data augmentation pipeline successfully generated a wide range of variations from the original 4D MRI volumes. By combining 3D volumetric augmentations with 2D slice-based augmentations, we significantly increased the diversity of the dataset.

2. Alignment and Accuracy:

One of the critical aspects of our pipeline was maintaining the alignment between the augmented images and their corresponding segmentation masks. This alignment was preserved throughout all augmentation steps, ensuring that the augmented data remained clinically relevant.

3. Visual Verification:

We employed our custom visualization function and NiLearn to visually inspect the augmented images. This step confirmed that the augmentations did not distort the anatomical structures and preserved the clinical features necessary for accurate segmentation.

4. Future Work:

While our current study focused on the development and verification of the augmentation pipeline, the diagnostic accuracy metrics, such as specificity and sensitivity, will be evaluated in future studies. These metrics will be essential in assessing the impact of the augmented data on the performance of deep learning models.

Conclusion:

The pipeline efficiently enhances small medical imaging datasets, creating diverse and clinically relevant data for future deep learning models.

Our study demonstrates that an efficient data augmentation pipeline can significantly enhance small medical imaging datasets by generating diverse and clinically relevant training samples. The augmentation techniques applied in both 3D and 2D spaces ensure that the augmented data retains anatomical accuracy and clinical significance.

This pipeline provides a scalable solution that can be integrated into various deep learning workflows, facilitating the development of robust and generalizable models. Future work will focus on training and benchmarking deep learning models using the augmented data to evaluate the impact on diagnostic performance.

The methodologies and findings from this study contribute to the broader field of medical imaging and deep learning, offering a valuable tool for researchers and practitioners working with limited datasets.

References:

1. Antonelli, M., Reinke, A., Bakas, S., et al. (2021). *The medical segmentation decathlon* (v1). arXiv. https://doi.org/10.48550/arXiv.2106.05735

2. Pérez-García, F., Sparks, R., & Ourselin, S. (2021). TorchIO: A Python library for efficient loading, preprocessing, augmentation, and patch-based sampling of medical images in deep learning. *Computer Methods and Programs in Biomedicine*, 206, 106236. https://doi.org/10.1016/j.cmpb.2021.106236

3. Kebaili, A.; Lapuyade-Lahorgue, J.; Ruan, S. (2023). Deep Learning Approaches for Data Augmentation in Medical Imaging: A Review. *Journal of Imaging*, 9(4), 81. https://doi.org/10.3390/jimaging9040081

4. NiBabel: https://nipy.org/nibabel/

5. PyTorch: https://pytorch.org/

6. NiLearn: https://nilearn.github.io/

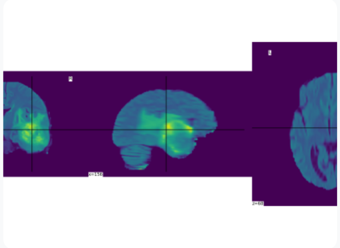


Fig 1: scan1_slice

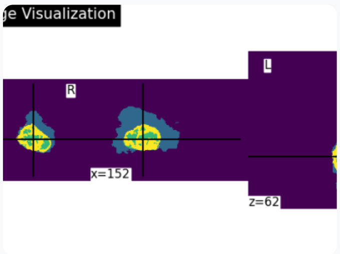


Fig 2: scan1_segmentation_mask

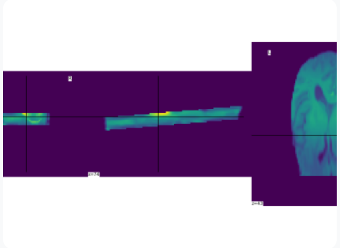


Fig 3: scan1_slice_transform1

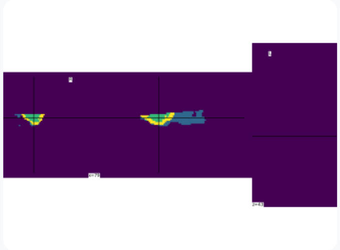


Fig 4: scan1_slice_segmentation_mask_transform1

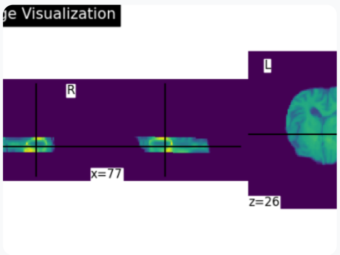


Fig 14: scan1_slice_transform2

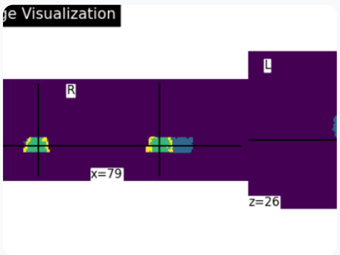


Fig 15: scan1_segmentation_mask_transform2

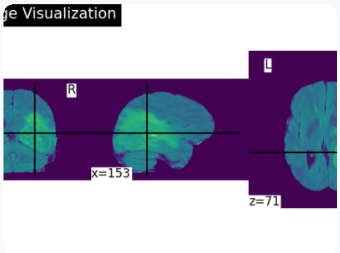


Fig 5: scan2_slice

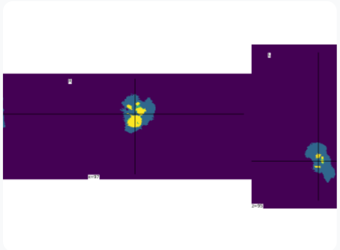


Fig 6: scan2_segmentation_mask

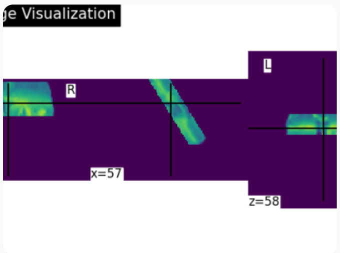


Fig 7: scan2_scan_slice_transform1

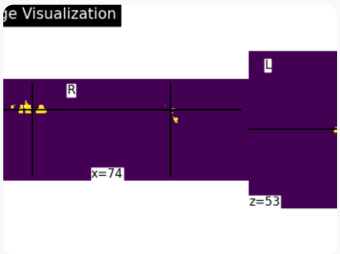


Fig 8: scan2_segmentation_mask_transform1

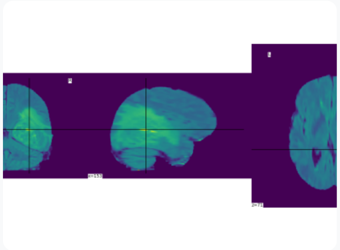


Fig 9: scan3_slice

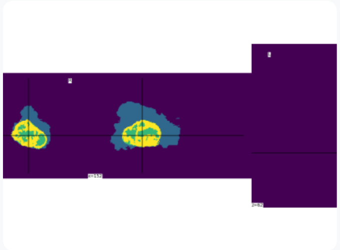
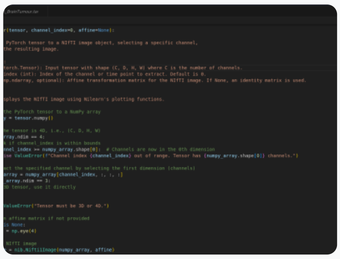
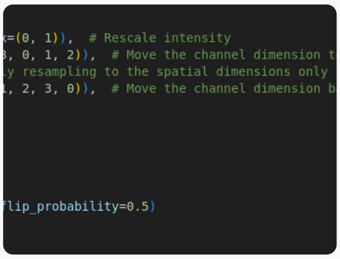


Fig 10: scan3_segmentation_mask



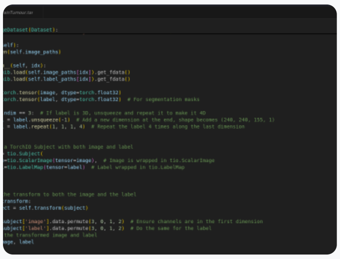
```
r(tensor, channel_index=0, affine=None):

 PyTorch tensor to a NIfTI image object, selecting a specific channel,
the resulting image.

:
torch.Tensor): Input tensor with shape (C, D, H, W) where C is the number of channels.
index (int): Index of the channel or time point to extract. Default is 0.
np.ndarray, optional): Affine transformation matrix for the NIfTI image. If None, an identity matrix is used.


splays the NIfTI image using Nilearn's plotting functions.

the PyTorch tensor to a NumPy array
y = tensor.numpy()

he tensor is 4D, i.e., (C, D, H, W)
rray.ndim == 4:
k if channel_index is within bounds
nnel_index >= numpy_array.shape[0]:  # Channels are now in the 0th dimension
ise ValueError(f"Channel index {channel_index} out of range. Tensor has {numpy_array.shape[0]} channels.")

act the specified channel by selecting the first dimension (channels)
array = numpy_array[channel_index, :, :, :]
_array.ndim == 3:
3D tensor, use it directly


ValueError("Tensor must be 3D or 4D.")

n affine matrix if not provided
is None:
 = np.eye(4)

 NIfTI image
 = nib.Nifti1Image(numpy_array, affine)
```

Fig 11: Code_plot_function



```
x=(0, 1)),  # Rescale intensity
3, 0, 1, 2)),  # Move the channel dimension t
ly resampling to the spatial dimensions only
1, 2, 3, 0)),  # Move the channel dimension b





flip_probability=0.5)
```

Fig 12: Code_train_transform



```
eDataset(Dataset):

elf):
n(self.image_paths)

__(self, idx):
ib.load(self.image_paths[idx]).get_fdata()
ib.load(self.label_paths[idx]).get_fdata()

orch.tensor(image, dtype=torch.float32)
orch.tensor(label, dtype=torch.float32)  # For segmentation masks

ndim == 3:  # If label is 3D, unsqueeze and repeat it to make it 4D
 = label.unsqueeze(-1)  # Add a new dimension at the end, shape becomes (240, 240, 155, 1)
 = label.repeat(1, 1, 1, 4)  # Repeat the label 4 times along the last dimension


a TorchIO Subject with both image and label
 tio.Subject(
=tio.ScalarImage(tensor=image),  # Image is wrapped in tio.ScalarImage
=tio.LabelMap(tensor=label)  # Label wrapped in tio.LabelMap



he transform to both the image and the label
ransform:
ct = self.transform(subject)

ubject['image'].data.permute(3, 0, 1, 2)  # Ensure channels are in the first dimension
ubject['label'].data.permute(3, 0, 1, 2)  # Do the same for the label
 the transformed image and label
age, label
```

Fig 13: Code_dataloader