A Hitchhiker's Guide to 3D Medical Image Processing (Preprocessing, Augmentations &

Toufiq Musah 10 min read · Just now




> *In this article, we will learn the 'what', 'why' and 'how' of 3D medical image processing.*

Starting our journey in deep learning and computer vision in medical imaging, we most likely encounter 2D images. On finding ourselves here, it is time to learn how to handle 3D medical images, as most real-world medical imaging cases are volumetric in nature.

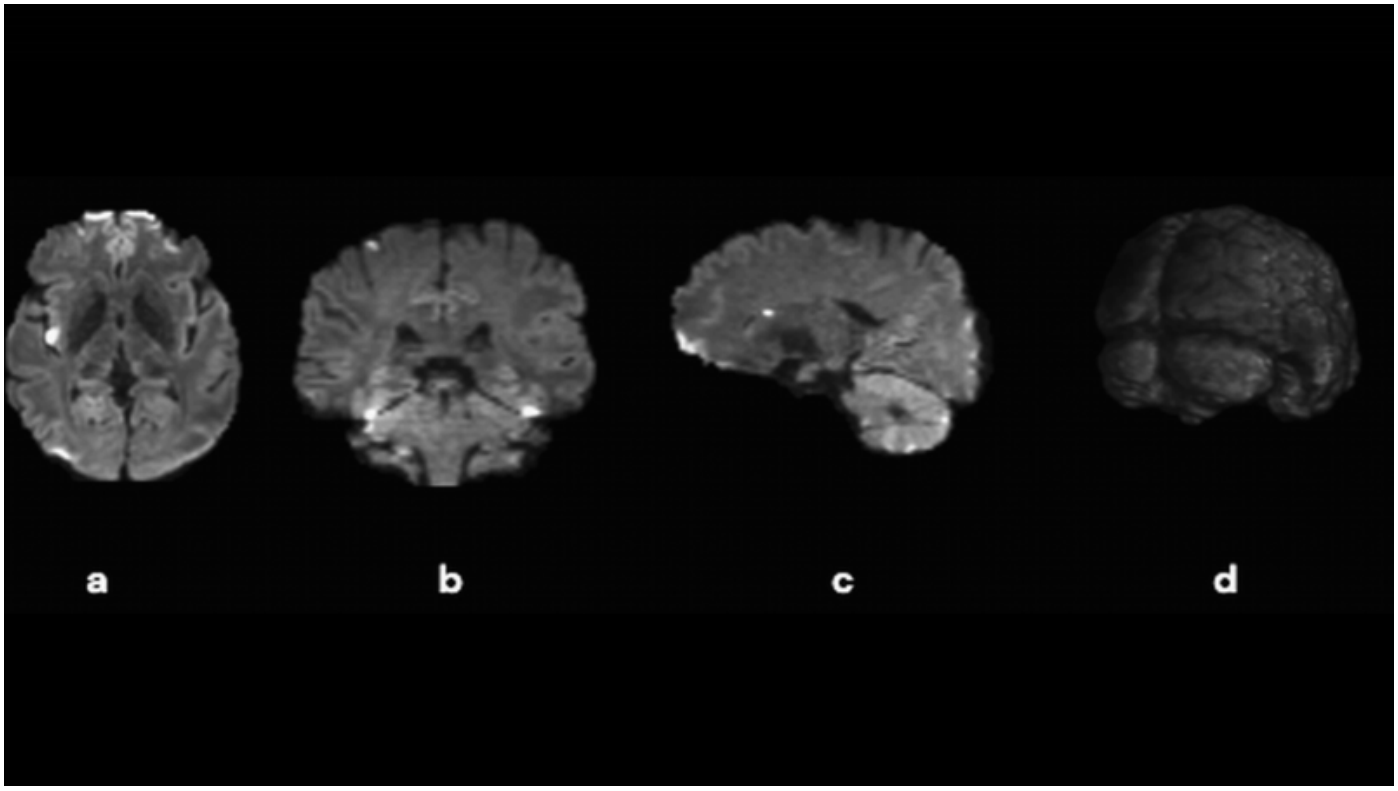


Fig. 1: 3D Brain Magnetic Resonance Image [1] (a) Axial View (b) Coronal View © Sagittal View (d) 3D Volumetric View

What are 3D Medical Images?

3D Medical Images are usually generated by radiological scans like **Computed Tomography**, & **Magnetic Resonance Imaging.** They are three-dimensional representations of the body's internal structures, capturing volumetric data, which allows viewing of anatomy from any angle. The most common viewing planes are the **axial** (fig. 1(a) top & bottom), **coronal** (fig. 1(b) front & back), and **sagittal** (fig. 1(c) left & right) views.

How Do they Differ from 2D? A 2D image is composed of a grid of **pixels**, each representing a single point in a two-dimensional plane. They are flat

and only provide information about a single projection of an object. Think of an x-ray, which is a single projection of a 3D object onto a 2D plane.

3D images on the other hand are composed of grids of voxels, representing points in three-dimensional space, with depth and volume. A 3D volume can be visualized as a stack of many 2D slices where each slice represents a specific cross-section of the anatomy (fig. 1(a-c)).

Voxel Spacing

It is important to understand voxel spacing in terms of 3D imaging. Voxel spacing describes the physical size of each voxel in millimeters along the three axes (x, y, z). This parameter determines the resolution of the image and influences how accurately anatomical structures are represented. For example, a scan with voxel spacing of **1 mm × 1 mm × 1 mm** means each voxel represents a cube of tissue measuring one millimeter on each side.

3D Medical Image Modalities

3D Medical Images come from different radiological techniques relying on different principles of physics to capture anatomy or function. The most common 3D modalities include CT, MRI, PET, and Ultrasound.

Computed Tomography (CT)

CT images are acquired using X-rays projected from multiple angles or sources, which are then reconstructed into slices. It is used for

anatomical imaging of both soft and hard tissues, including the brain, bones, lungs, and cardiovascular system. CT is fast and useful in emergency diagnostics, though it does expose patients to ionizing radiation.

Magnetic Resonance Imaging (MRI)
MRIs are generated by applying radio-frequency pulses that excite hydrogen atoms in the body and then measuring the returning signals. MRI does not use ionizing radiation. It is best at soft tissue imaging, which is why it is the modality of choice for brain and organ imaging. It comes in several flavors (sequences), each highlighting varying tissue properties (T1, T2, FLAIR, DWI, FLAIR).

Positron Emission Tomography (PET)
PET imaging is performed by injecting patients with radiotracers and measuring the radiation emitted by isotopes as they decay. PET provides functional information by highlighting areas of metabolic. It is commonly used in oncology for tumor detection, and in neurology for brain activity mapping. PET is often combined with CT or MRI to fuse structural and functional data.

Ultrasound (US)
Ultrasound uses high-frequency sound waves to produce images of internal structures based on echoing signals from different tissue densities. 3D Ultrasound Imaging is relatively uncommon, and is usually reserved for fetal or cardiac imaging for some suspected cases of anomalies.

3D Medical Image Formats

Medical images are usually stored alongside metadata such as scanner type & patient identifiers. Because of the unique nature of 3D Medical Images and the amount of other data they carry, standardized formats have been developed for different use cases. Most of them are interchangeable and can be adapted for analysis of various kinds.

Neuroimaging Informatics Technology Initiative (NIfTI)
The NIfTI format was originally proposed for storing neuroimaging data but has since become one of the most widely used formats in medical image analysis due to it simple and adaptable nature [4]. NIfTI files typically have the extensions `.nii` or `.nii.gz` and can be handled in Python using libraries such as **NiBabel**. A NIfTI file contains both the volumetric image data and structured headers describing dimensions, voxel spacing, orientation, and more. For example, `get_fdata()` in NiBabel provides direct access to the voxel data array.

Digital Imaging and Communications in Medicine (DICOM)
DICOM [5] is the most common format in clinical environments. A single 3D volume in DICOM is often stored as a collection of 2D slices, each with its own header, containing extensive metadata such as acquisition parameters, scanner details, and patient information. Working directly with DICOM can be hectic, ergo why researchers often convert it to formats like NIfTI for downstream analysis using the dicom2nifti library.

Nearly Raw Raster Data (NRRD)
NRRD format is designed for storing and sharing n-dimensional raster data, including 3D medical images. NRRD files usually come as `.nrrd` or `.nhdr` (separate data file) and can be read with Python libraries such as **pynrrd**. Compared to DICOM, NRRD is lightweight, making it suitable for applications simplicity is preferred, such as visualization.

> *We are done with the 'What' of Medical Imaging. Lets move on to the 'How'.*

The usual workflow in working with Medical Images for analysis begins with **preprocessing** the data to standardize it, followed by applying **augmentations** to improve model generalization, and finally preparing efficient **DataLoaders** to feed the data into the models.

To make this process easier, several open-source frameworks have been developed, with **MONAI** [2] and **TorchIO** [3] being among the most popular. Other useful tools include **SimpleITK** [6] for image preprocessing, **NiBabel** for working with NIfTI files, and **batchgenerators** [7], widely used in medical imaging challenges.

TorchIO for 3D Medical Imaging

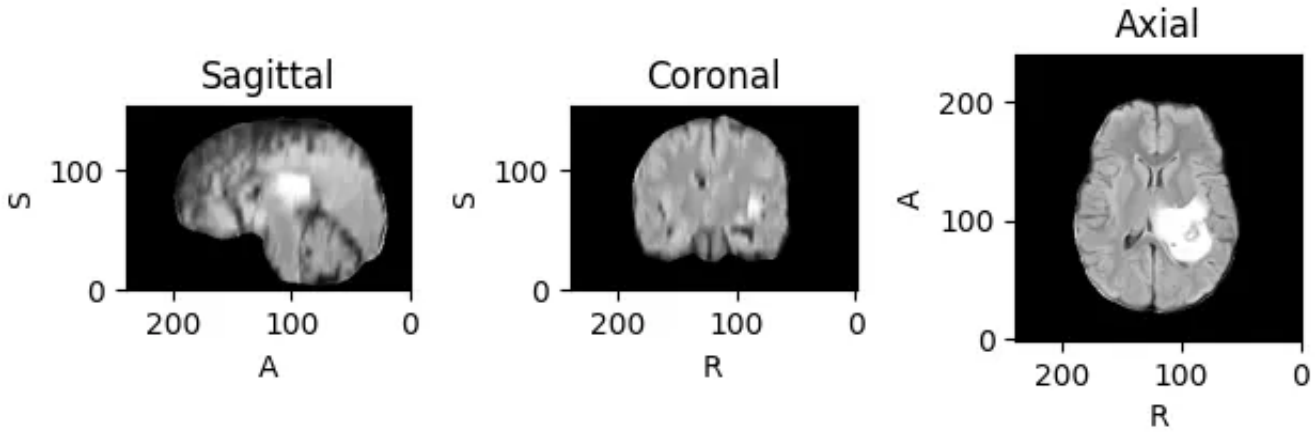
For the purposes of this guide, we will be using TorchIO to learn the 'How' of 3D Medical Imaging. We will be building on Parts 1 [8], & 2 [9] of previous work. We will begin with learning the basics of loading and preprocessing 3D Medical Imaging data, in this case, NIfTi **Multi-**

Parametric Brain MRI from the **Brain Tumor Segmentation Africa Dataset** [10], followed by augmentations and dataloaders. Let's get our hands dirty with some code.

Downloading and Loading Data

```
# library installs & dataset download

!pip -q install gdown
!pip -q install torchio

!gdown 1osj3lgzNlsW-esdq0KQvvLB7yiadyZGA
!unzip -q "BrainMRI-Samples.zip"

# importing libraries

import os
import shutil
import numpy as np
import matplotlib.pyplot as plt

import torchio as tio
```

```
# Loading and Viewing MRI Sample

mri_path = 'BraTS-SSA-Samples/BraTS-SSA-00007-000/BraTS-SSA-00007-000-t2f.ni
mri_sample = tio.ScalarImage(mri_path)

mri_sample.plot()
print("MRI Shape:",mri_sample.shape)
```



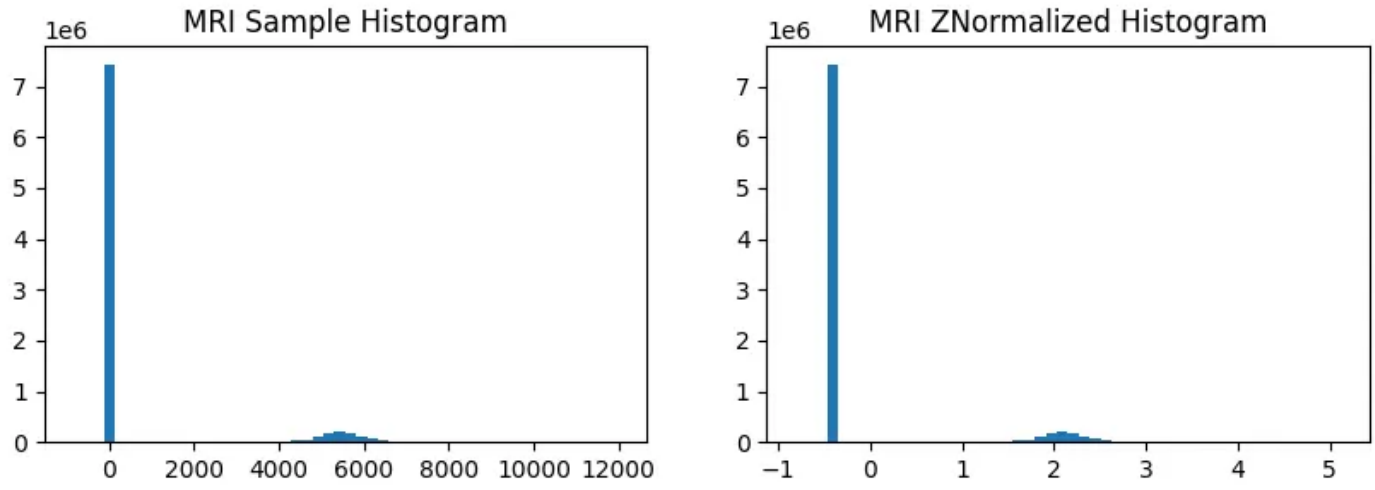
3D Medical Image Preprocessing

Preprocessing can be generally categorized into Intensity & Spatial transforms. Under Intensity transformations, we can modify MRI scan intensities through methods such as `rescaling` and `normalization`. Spatial transforms influence the physical structure of a the volume, with methods such as `cropping/padding`, and voxel space `resampling` (remember voxel spacing from earlier?)

Let's apply `ZNormalization`, a widely recommended standardization transform for medical images. This transformation standardizes pixel intensities by subtracting the mean and dividing by the standard deviation, ensuring that the data has a mean of zero and a unit variance. This helps improve model performance by making the data more consistent.

```
# Defining and Applying ZNormalization

z_transform = tio.ZNormalization()
mri_znormalized = z_transform(mri_sample)

plt.subplot(1, 2, 1)
plt.hist(mri_sample.data.flatten(), bins=50)
```

```
plt.title('MRI Sample Histogram')

# Plotting ZNormalized MRI Histogram
plt.subplot(1, 2, 2)
plt.hist(mri_znormalized.data.flatten(), bins=50)
plt.title('MRI ZNormalized Histogram')
```



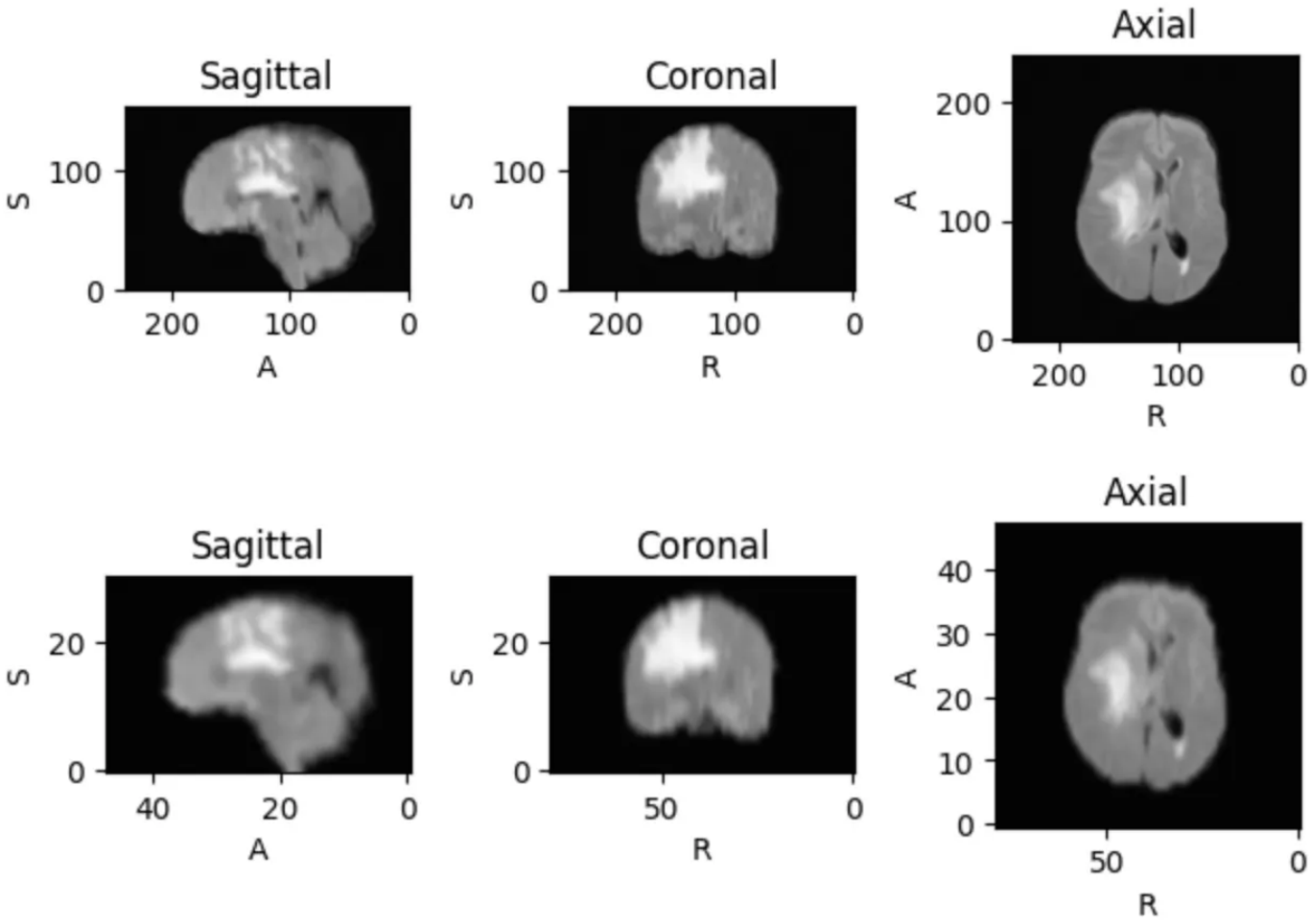
Histogram of Normalized vs. Unnormalized MRI (peep the x-axis)

Notice in the above histogram, the MRI range of intensities has shifted from 0 to 12000, to from -1 to 5, which is a more manageable range of values.

Up next lets try voxel space resampling, and then move on to augmentations.

```
# Defining traget_space of Width, Height and Depth
target_spacing = (3, 5, 5)

resample_transform = tio.Resample(target_spacing)

# Apply the resampling transform to your MRI sample
mri_resampled = resample_transform(mri_sample)
```

```
# Visualizing MRI (Top: Original Sample, Bottom: Resampled MRI
mri_sample.plot()
mri_resampled.plot()
print("Resampled shape:", mri_resampled.shape)
```



(Top) 1mm Uniform Voxel Spacing. (Bottom) Distorted Voxel Spacing

In the above, we massively distort the voxel spacing of the image from: *(1.0, 1.0, 1.0) to (1.5, 2.0, 5.0)*, leading to a much poorer quality image.

3D Medical Image Augmentations

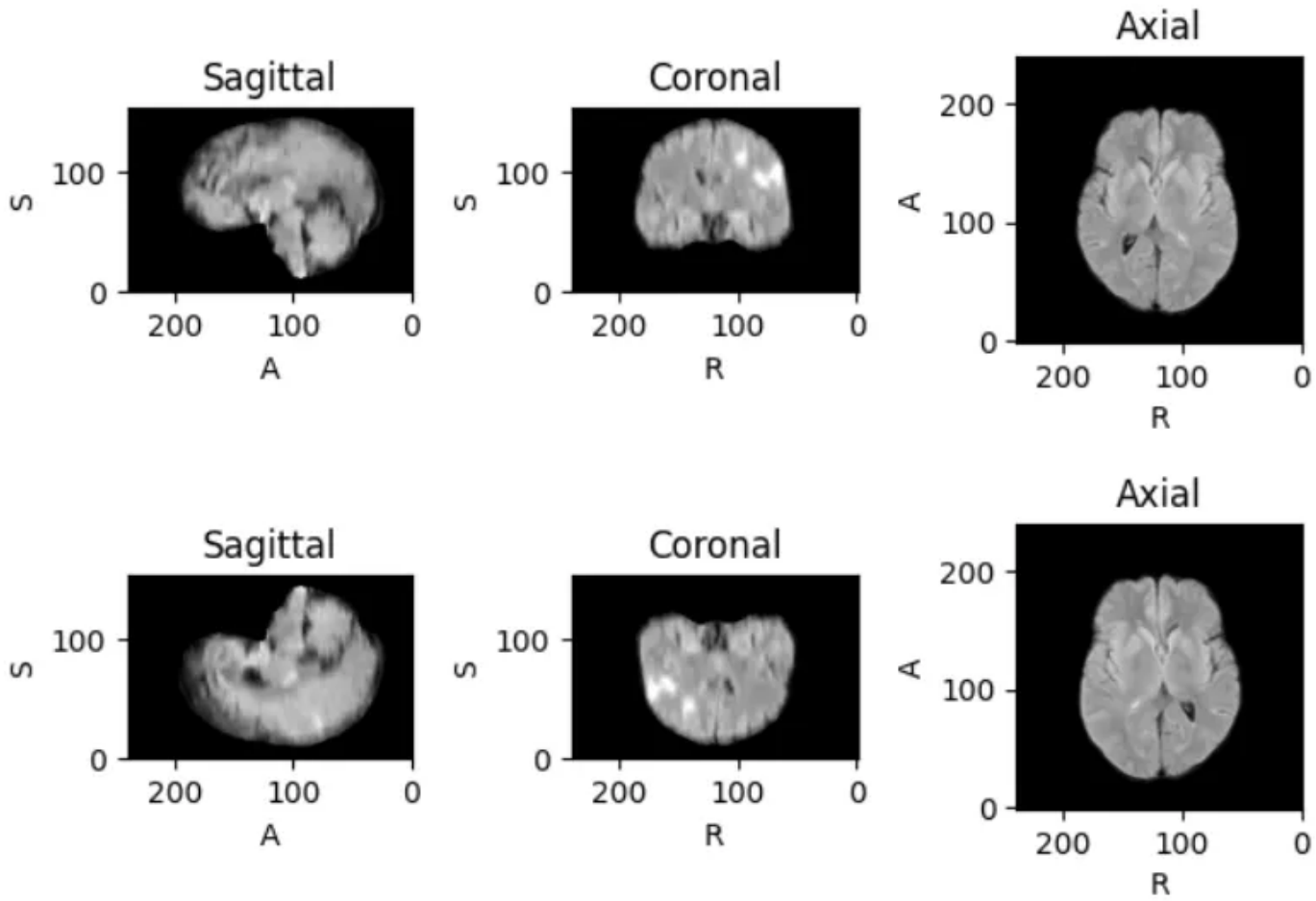
Augmentation transforms are important for creating variability in batches of data when training deep learning models. They introduce

noticeable alterations to both the spatial structure and intensity values of the images.

Spatial augmentations can be categorized as either affine or elastic. Affine transformations preserve points and planes within the image while introducing variations in positioning such as slight rotations, translations, and flips. Elastic transformations, on the other hand, apply non-linear deformations that simulate anatomical variability, such as differences in head shape.

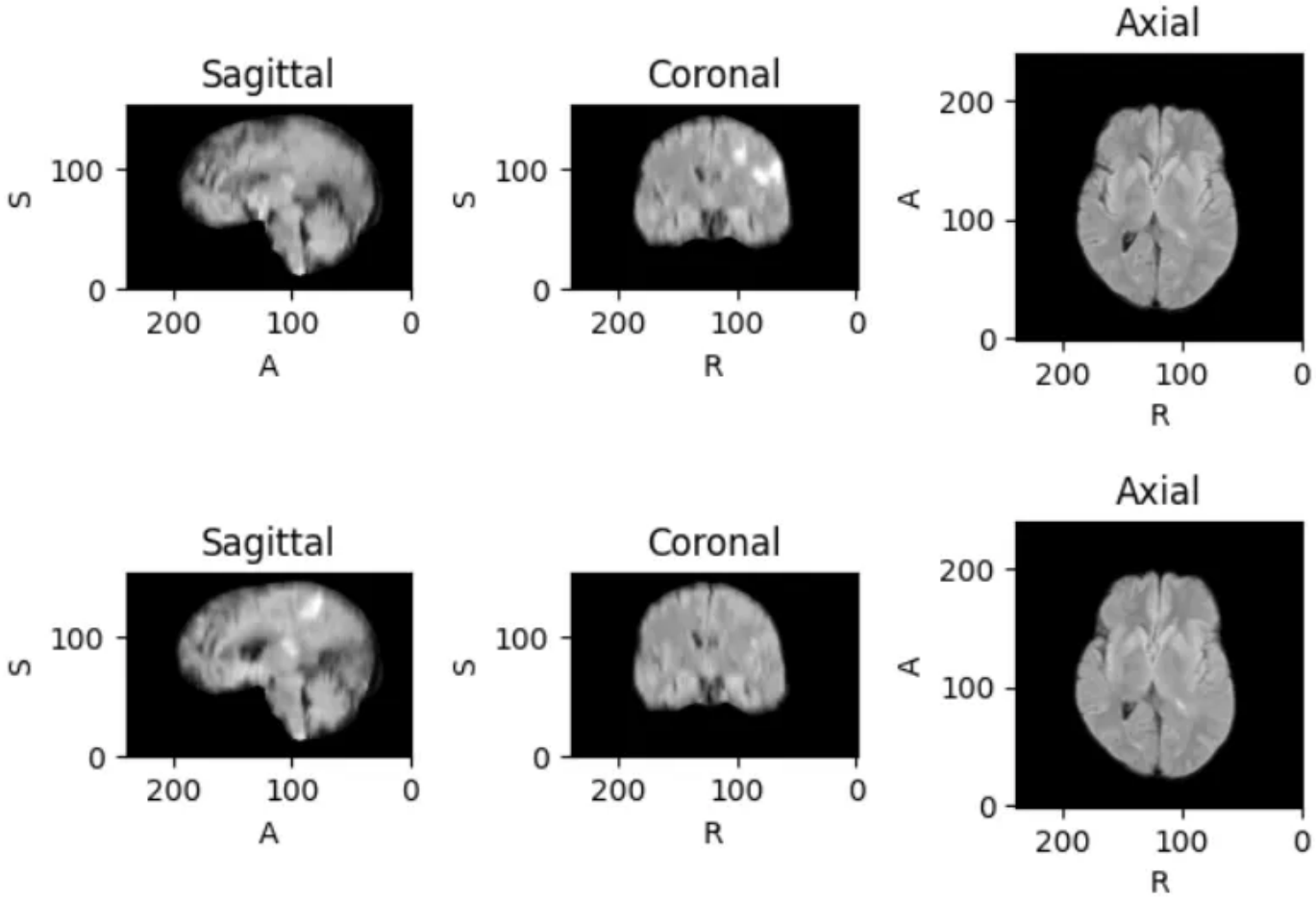
Lets try random flipping transformations first:

```
# Applying Random Flip to MRI

random_flip = tio.RandomFlip(p=0.9, axes=(0,1,2))
mri_flipped = random_flip(mri_sample)

mri_sample.plot()
mri_flipped.plot()
```



That.. is obvious. Lets try an elastic transform.

```
# Applying Random Elastic Transformations to MRI

random_elastic = tio.RandomElasticDeformation(p=0.9)
mri_elastic = random_elastic(mri_sample)

mri_sample.plot()
mri_elastic.plot()
```



(Top) Sample. (Bottom) Elastic Deformed Sample.

Notice the variations in the hyper-intensities at the top-right of the Coronal slice for example.

Okay, we are getting to the endgame now. Creating DataLoaders

DataLoaders in 3D Medical Imaging

The DataLoader handles the loading, application of transforms, and batching of the data and labels for deep learning model training.

What's the best way to apply several transforms to a single MRI sample? We often need to apply a sequence of transforms, such as **intensity normalization** and **spatial resampling.** The best way to chain transforms

like this is by using the `tio.Compose` method. Let's create a a custom **SegmentationDataset** loader using TorchIO.

```
# imports and all ..

import os
from glob import glob

import torchio
from torchio.data import SubjectsLoader, SubjectsDataset
```

We'll now create a custom dataset class. This class will handle loading each MRI sample (along with its modalities) and the corresponding segmentation label. Each sample will be wrapped as a `TorchIO.Subject`, which conveniently stores the image and label as named components, which we can later access using dictionary-style keys.

```
# Custom Dataset - 🧠

class SegmentationDataset(SubjectsDataset):
    def __init__(self, root_dir: str, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        # Recursively Finding MRI and Segmentation Label Paths with Glob
        self.t2f_paths = sorted(glob(os.path.join(root_dir, "**", "*t2f*.nii
        self.t1c_paths = sorted(glob(os.path.join(root_dir, "**", "*t1c*.nii
        self.seg_paths = sorted(glob(os.path.join(root_dir, "**", "*seg*.nii
        self.subjects = [
            tio.Subject(
                mri=tio.ScalarImage([t1c_path, t2f_path]),
                label=tio.LabelMap(seg_path)
            )
            for t2f_path, t1c_path, seg_path in zip(self.t2f_paths, self.t1c_
        super().__init__(self.subjects, transform=transform)
```

We use `glob` to recursively find the indicated modalities, and load them into a `tio.Subject` object. When the `SegmentationDataset` class is called, it returns a Subject containing the modalities, `t1c` (T1 Contrast) and `t2f` (T2-Flair), and the segmentation map, `label`.

Note that we only use 2 BraTS modalities as an example. Give it a try and load all 4 modalities (including `t1n`, and `t2w`), okay?

Lets now use tio.Compose to create a chain of preprocessing and augmentation steps our data will undergo in a DataLoader.

```
# Preprocessing Transforms:

preprocessing_transforms = tio.Compose(
    [
        tio.RescaleIntensity(out_min_max=(0, 99.5), include = ['mri']),
        tio.ZNormalization(include = ['mri']),
        tio.Resample((2, 2, 2), include = ['mri', 'label']),
        tio.CropOrPad((96, 96, 64), include = ['mri', 'label']),
        tio.OneHot(num_classes = 4, include = ['label']),
    ]
)

# Augmentation Transforms:
augmentation_transforms = tio.Compose(
    [
        tio.RandomFlip(axes=(0,1,2), p=0.5, include = ['mri', 'label']),
        tio.RandomAffine(p=0.5, include = ['mri', 'label']),
        tio.RandomElasticDeformation(p=0.5, include = ['mri', 'label']),
    ]
)

# All the Transforms
data_transforms = tio.Compose(
```

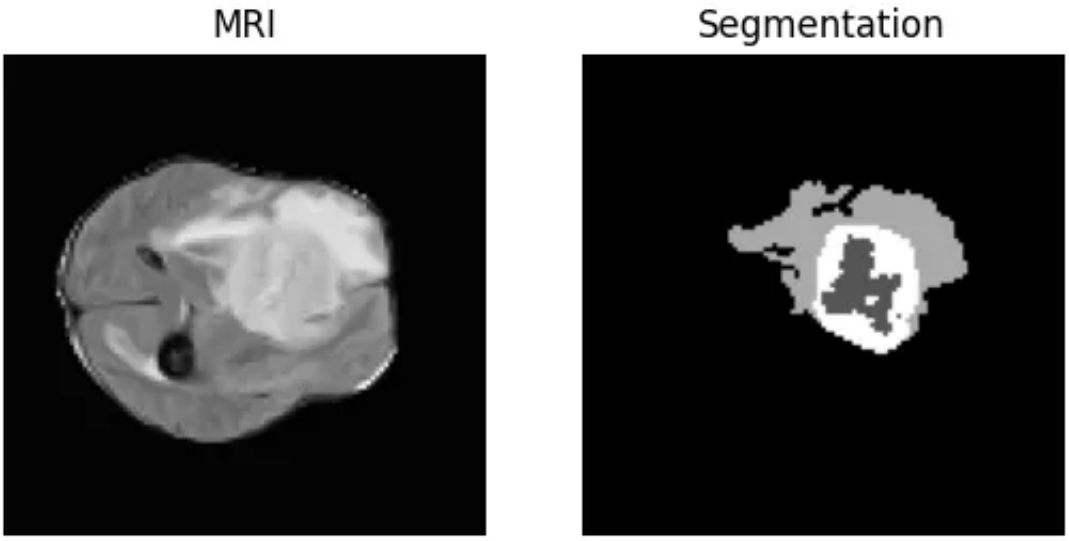
```
        [
            preprocessing_transforms,
            augmentation_transforms
        ]
    )
```

Notice that we introduce a new `include` parameter which tells what sample item in the dataset should be affected by a transform, and only include the labels for transforms that geometrically alter the MRI, such as `tio.Resample`, `tio.RandomFlip`, but not `tio.OneHot`. If a transform may affect class specific pixels (such as noise), then labels should be excluded from that transform.

We further introduce a new preprocessing step, `tio.OneHot()`, and exclusively apply that to the labels. We set `num_classes` to 4, representing the class label values in the BraTS dataset with *0 — background, 1 — Tumor Core, 2 — Whole Tumor, 3 — Enhancing Tumor.* (This may be unnecessary based on your training pipeline)

Now let's instantiate our `SubjectDataset` class, and then create a TorchIO `SubjectsLoader` for data loading.

```
# dataloader, etc

dataset = SegmentationDataset('BraTS-SSA-Samples/', transform=data_transforms
dataloader = SubjectsLoader(dataset, batch_size=4, shuffle=True)
batch = next(iter(dataloader))
mri_data = batch['mri'][tio.DATA]
seg_data = batch['label'][tio.DATA]
print(mri_data.shape, seg_data.shape)
```



The end.. *Thanks for coming to my TED Talk.*

Complementary Notebooks

1. Introduction to TorchIO (Part 1)
2. Introduction to TorchIO (Part 2)

References

[1] Hernandez Petzsche *et al.* ISLES 2022: https://doi.org/10.1038/s41597-022-01875-5

[2] MONAI: An open-source framework for deep learning in healthcare

[3] TorchIO: A Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images

[4] NIfTI: https://nifti.nimh.nih.gov/

[5] DICOM: https://www.dicomstandard.org/

[6] SimpleITK: SimpleITK

[7] Batchgenerators: https://github.com/MIC-DKFZ/batchgeneratorsv2

[8] Introduction to TorchIO (Part 1)

[9] Introduction to TorchIO (Part 2)

[10] Adewole, Maruf, et al. "The BraTS-Africa Dataset: Expanding the Brain Tumor Segmentation Data

Medical Imaging | Deep Learning | Computer Vision | Dataloader

Written by Toufiq Musah

20 followers · 16 following

Edit profile

```
# Acquaring the sample dataset

# !pip -q install gdown

!gdown 1-_a81TB5UQhpLgyTwTVKADikESOLWl7C
!unzip -q "BrainMRI-Samples.zip"
```

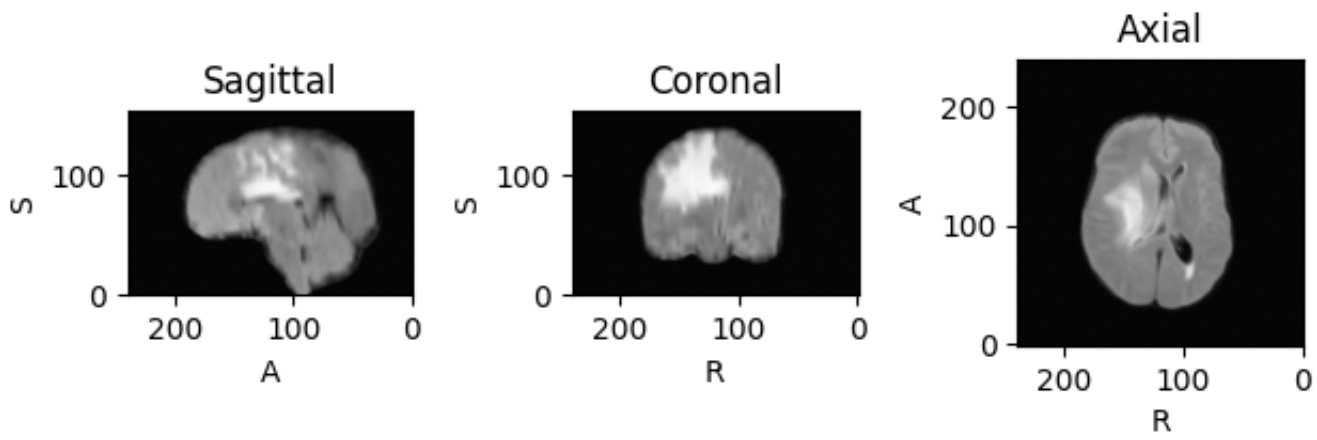
```
Downloading...
From: https://drive.google.com/uc?id=1-_a81TB5UQhpLgyTwTVKADikESOLWl7C
To: /content/BrainMRI-Samples.zip
100% 19.3M/19.3M [00:00<00:00, 60.2MB/s]
```

```
# Installing Libraries

!pip -q install torchio
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.1/53.1 kB 3.4 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.2/194.2 kB 11.3 MB/s eta 0:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.6/52.6 MB 13.5 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 2.8 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 108.0 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 77.1 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 41.4 MB/s eta 0:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.0 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 6.2 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 16.4 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.1 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 5.8 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 188.7/188.7 MB 5.8 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 53.4 MB/s eta 0:00
```

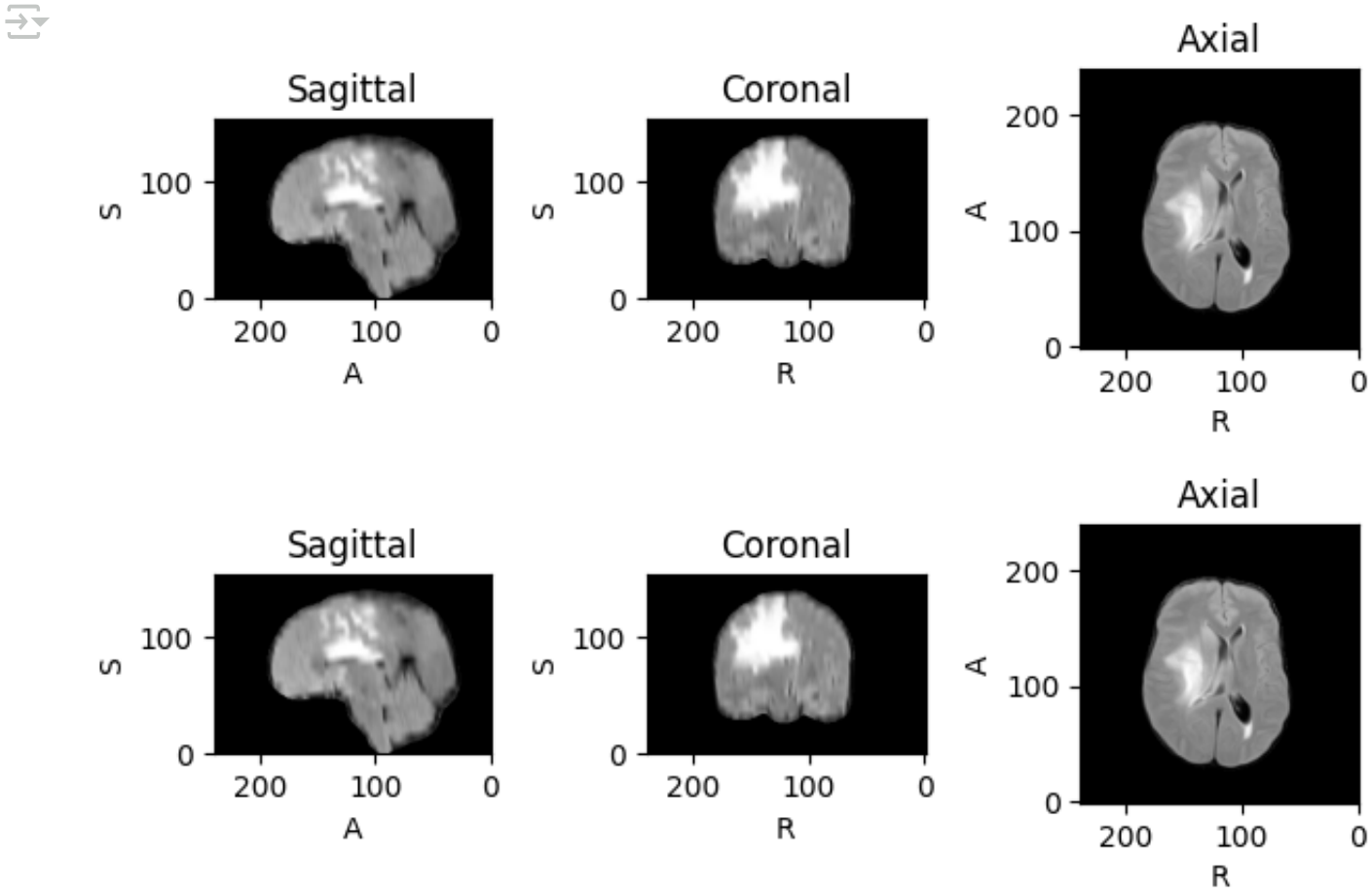
```
# Importing Libraries...

import os
import shutil

import numpy as np
import matplotlib.pyplot as plt

import torchio as tio
```

```
# Loading and Viewing MRI Sample

mri_path = '/content/BraTS-SSA-00002-000-t2f.nii.gz'

mri_sample = tio.ScalarImage(mri_path)
mri_sample.plot()
print("MRI Shape:",mri_sample.shape)
```



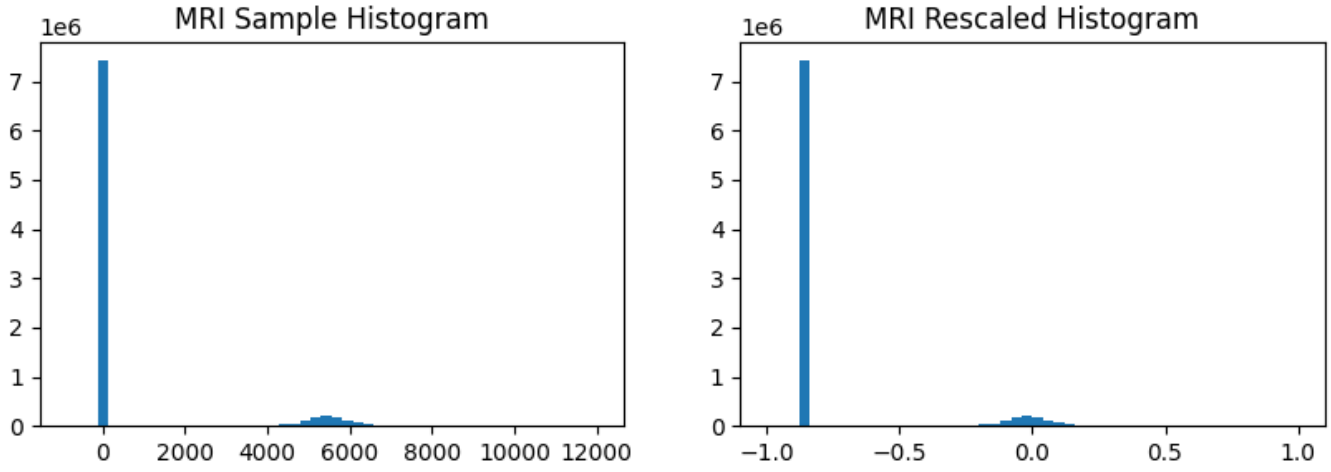

```
MRI Shape: (1, 240, 240, 155)
```

```
 # Intensity Rescaling

min_val = -1
max_val = 1

# Defining and Applying Intensity Rescaler
rescaler = tio.RescaleIntensity((min_val, max_val))
mri_rescaled = rescaler(mri_sample) # Applying intensity rescaler

# Visualizing MRI (Top: Original Sample, Bottom: Rescaled MRI)
mri_sample.plot()
mri_rescaled.plot()
```

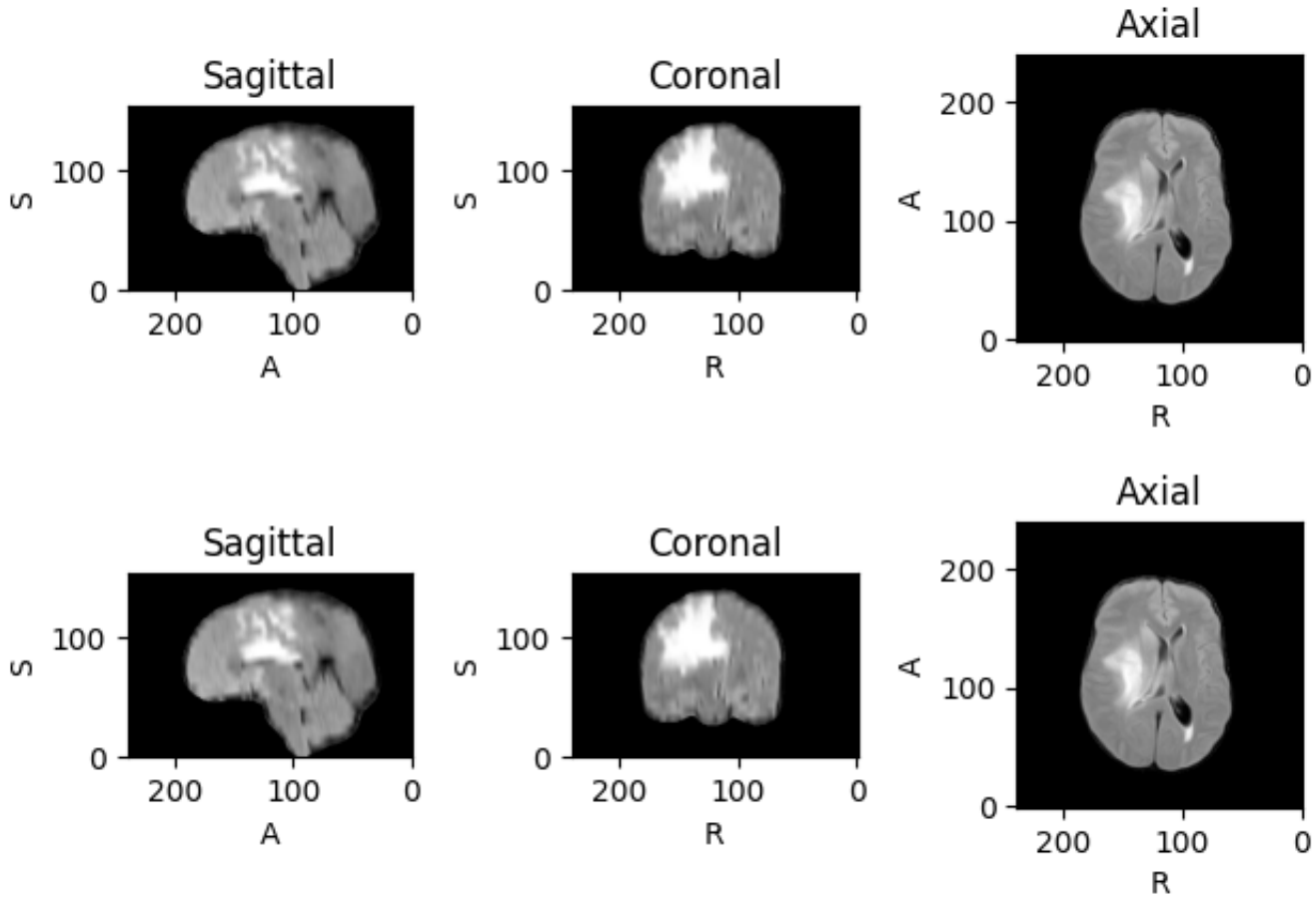


```
# Plotting MRI Sample Histogram
plt.figure(figsize=(10, 3))

plt.subplot(1, 2, 1)
plt.hist(mri_sample.data.flatten(), bins=50)
plt.title('MRI Sample Histogram')

# Plotting Rescaled MRI Histogram
plt.subplot(1, 2, 2)
plt.hist(mri_rescaled.data.flatten(), bins=50)
plt.title('MRI Rescaled Histogram')
```

```
Text(0.5, 1.0, 'MRI Rescaled Histogram')
```

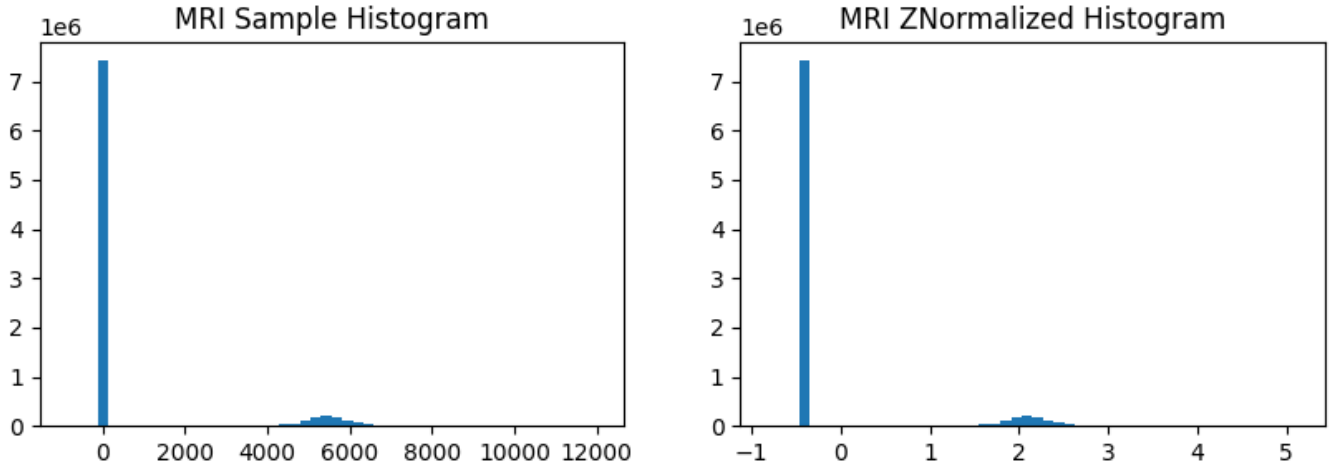


```
# Defining and Applying ZNormalization

z_transform = tio.ZNormalization()
mri_znormalized = z_transform(mri_sample)

# Visualizing MRI (Top: Original Sample, Bottom: ZNormalized MRI)
mri_sample.plot()
mri_znormalized.plot()
```

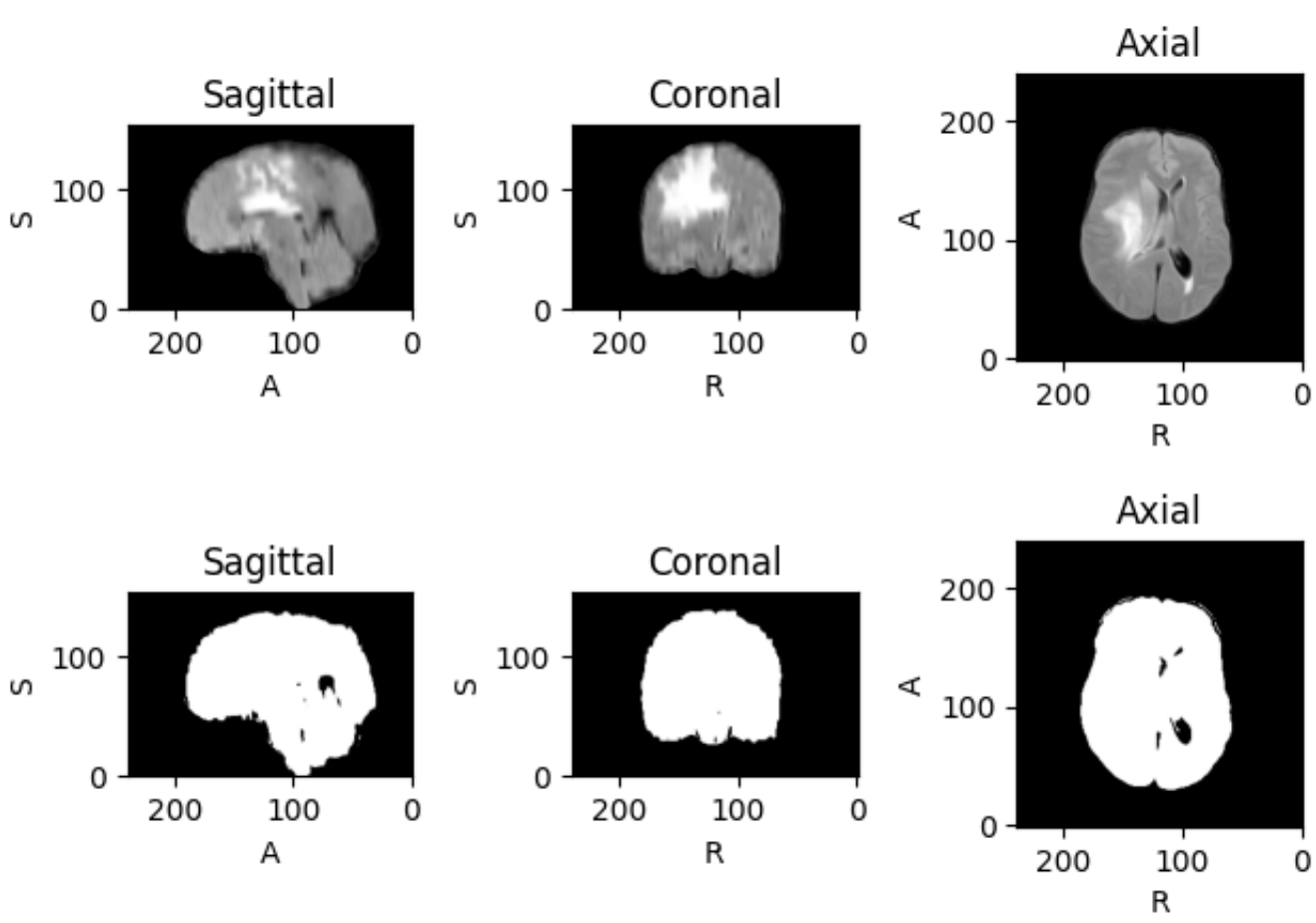


```
# Plotting MRI Sample Histogram
plt.figure(figsize=(10, 3))

plt.subplot(1, 2, 1)
plt.hist(mri_sample.data.flatten(), bins=50)
plt.title('MRI Sample Histogram')

# Plotting ZNormalized MRI Histogram
plt.subplot(1, 2, 2)
plt.hist(mri_znormalized.data.flatten(), bins=50)
plt.title('MRI ZNormalized Histogram')
```

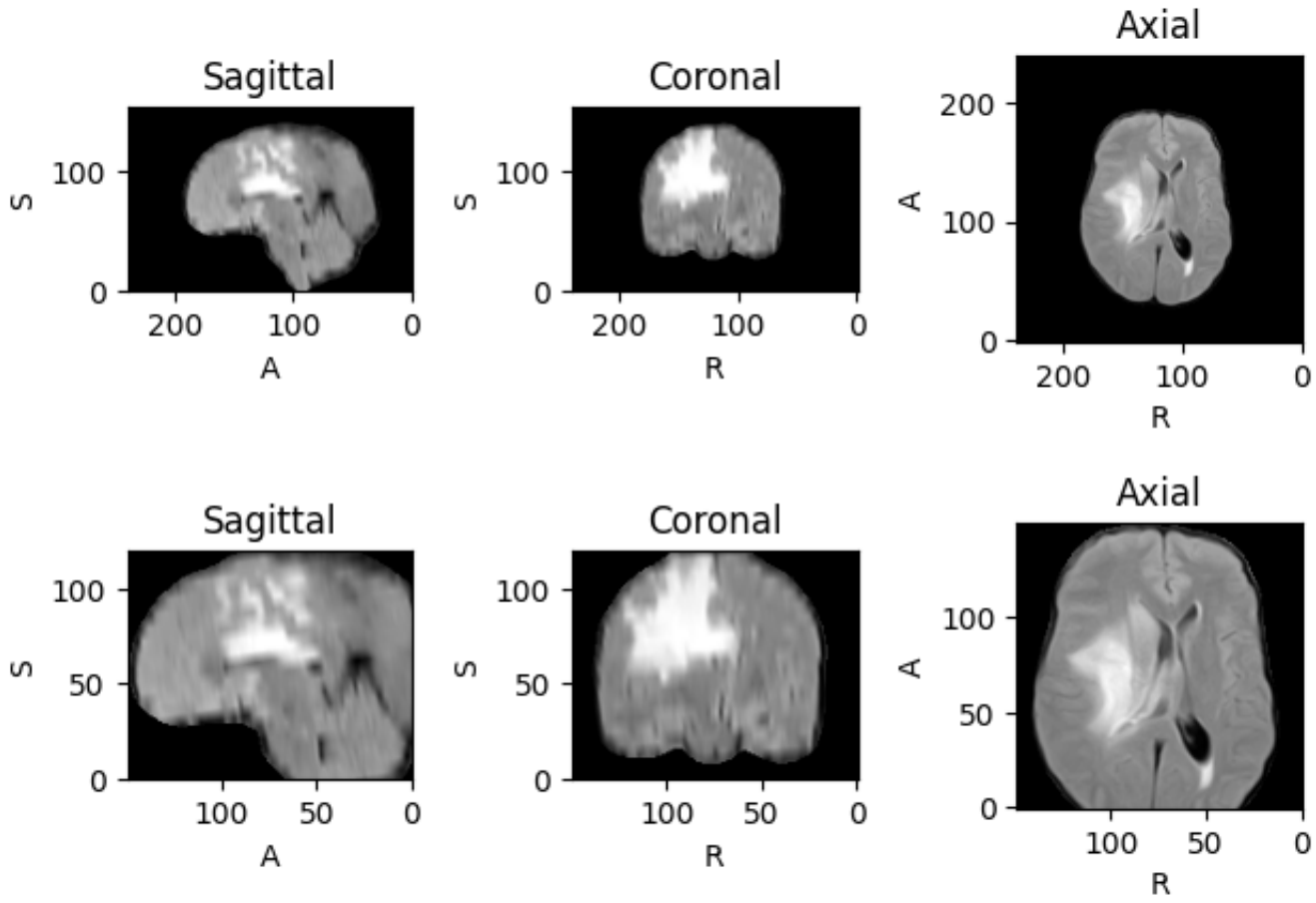
```
Text(0.5, 1.0, 'MRI ZNormalized Histogram')
```



```
# Intensity Clamping Transforms

min_val = 2500
max_val = 3000
clamp_transform = tio.Clamp(min_val, max_val)
mri_clamped = clamp_transform(mri_sample)

# Visualizing MRI (Top: Original Sample, Bottom: Clamped MRI)
mri_sample.plot()
mri_clamped.plot()
```

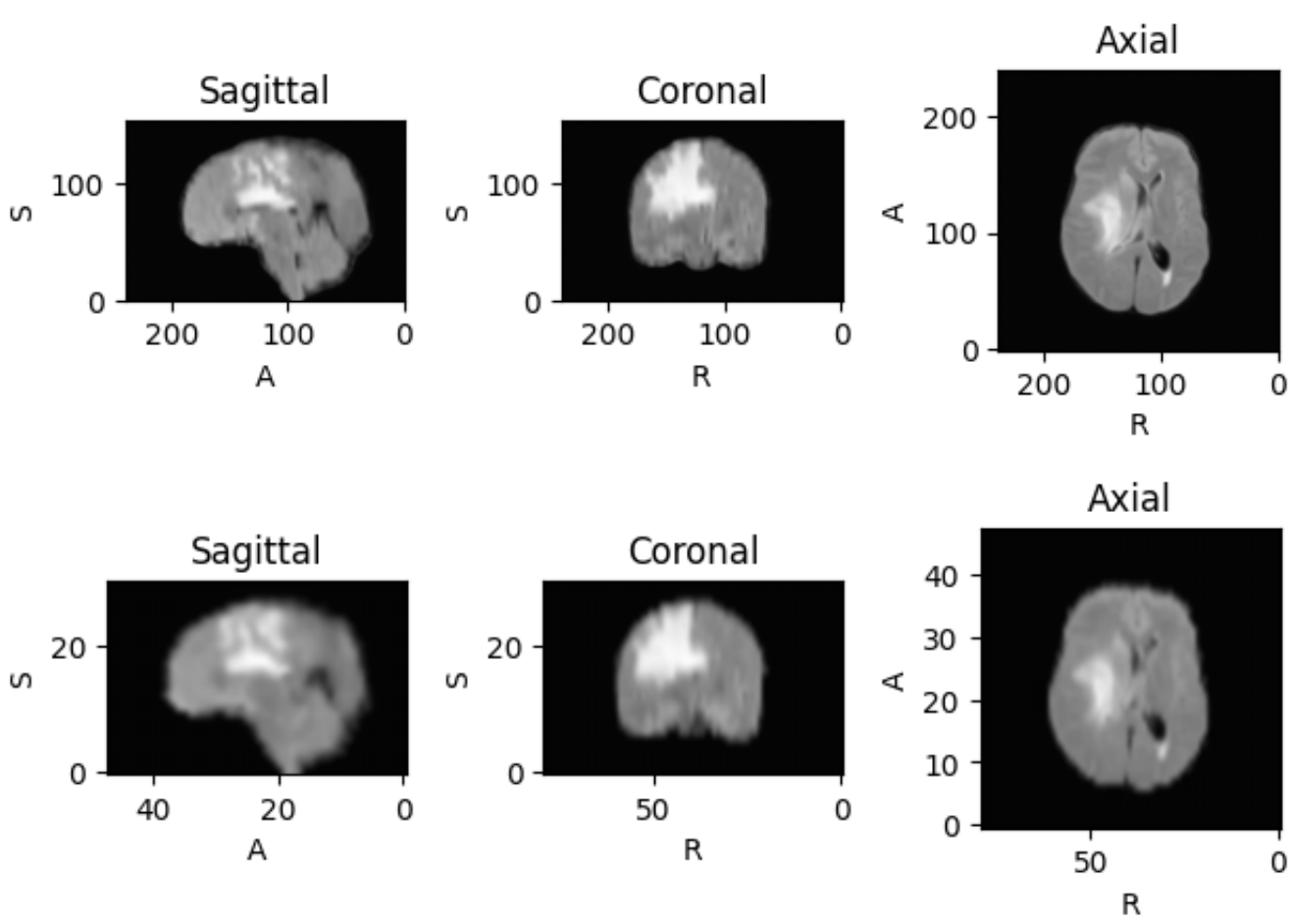


```
# Defining traget_shape with Width, Height and Depth
target_shape = (150, 150, 120)

# Defining and Applying Crop
crop_transform = tio.CropOrPad(target_shape)

mri_cropped = crop_transform(mri_sample)

# Visualizing MRI (Top: Original Sample, Bottom: Cropped MRI)
mri_sample.plot()
mri_cropped.plot()
mri_cropped.shape
```



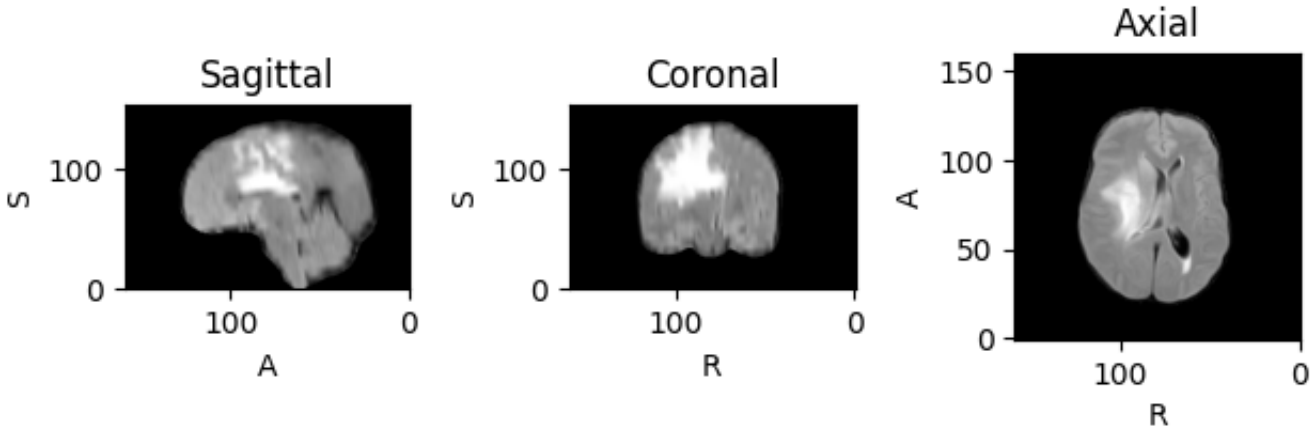

```
(1, 150, 150, 120)
```

```
# Defining traget_space of Width, Height and Depth
target_spacing = (1.5, 1.5, 1)
target_spacing = (3, 5, 5)

resample_transform = tio.Resample(target_spacing)

# Apply the resampling transform to your MRI sample
mri_resampled = resample_transform(mri_sample)

# Visualizing MRI (Top: Original Sample, Bottom: Resampled MRI
mri_sample.plot()
mri_resampled.plot()
print("Resampled shape:", mri_resampled.shape)
```




```
Resampled shape: (1, 80, 48, 31)
```

```
# applying several transforms

target_spacing = (1.5, 1.5, 1)

# Define transformation pipeline
preprocess_pipeline = tio.Compose([
    tio.ZNormalization(),
    tio.Resample(target_spacing)
])

# Apply the composed transforms to the MRI sample
mri_transformed = preprocess_pipeline(mri_sample)
mri_transformed.plot()
```



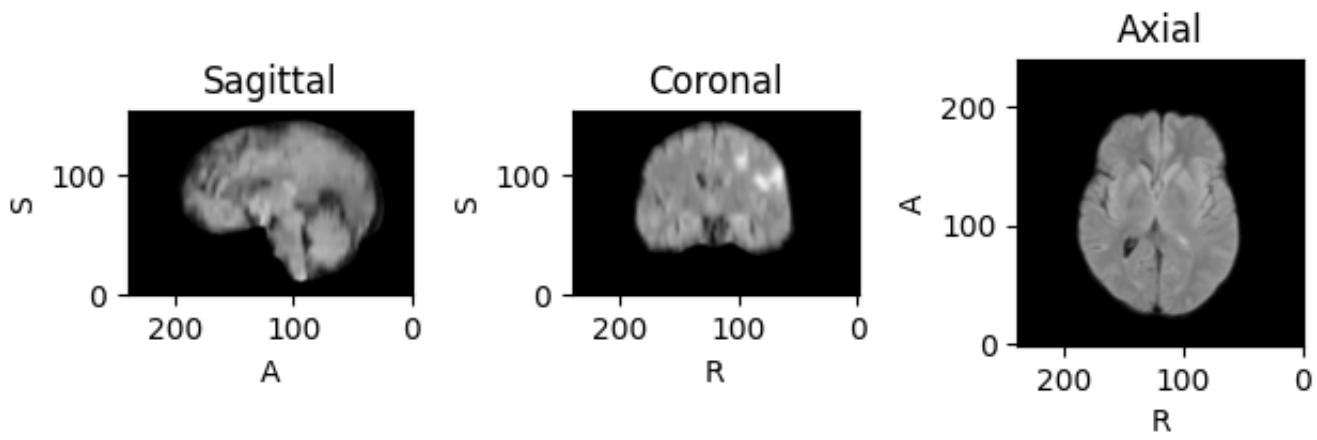
```
# library installs & dataset download

# !pip -q install gdown
!pip -q install torchio

!gdown -q 1osj3lgzNlsW-esdq0KQvvLB7yiadyZGA
!unzip -q "/content/BraTS-SSA-Samples.zip" -d "/content/BraTS-SSA-Samples"
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.1/53.1 kB 2.0 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.2/194.2 kB 7.2 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.6/52.6 MB 8.0 MB/s eta 0:00:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.1 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 41.9 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 40.8 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 30.0 MB/s eta 0:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.1 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.4 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 14.6 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 8.1 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.2 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 188.7/188.7 MB 6.4 MB/s eta 0:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 64.0 MB/s eta 0:00
replace /content/BraTS-SSA-Samples/BraTS-SSA-00213-000/BraTS-SSA-00213-000-
```

```
# Importing Libraries...

import os
import shutil

import numpy as np
import matplotlib.pyplot as plt

import torchio as tio
```

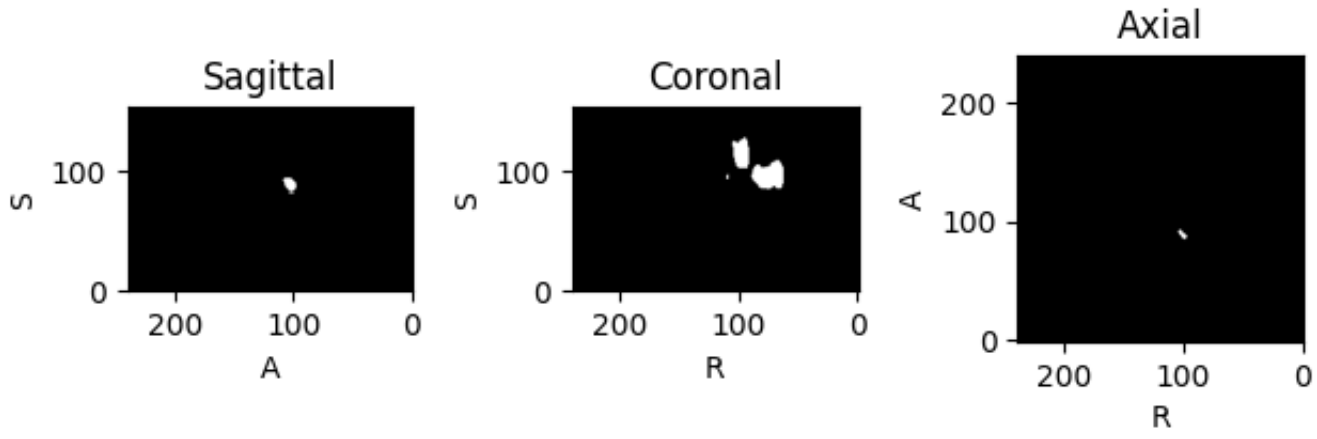
Augmentations

```
# Loading and Viewing MRI Sample

mri_path = '/content/BraTS-SSA-Samples/BraTS-SSA-00049-000/BraTS-SSA-00049-000-

mri_sample = tio.ScalarImage(mri_path)
mri_sample.plot()
print("MRI Shape:",mri_sample.shape)
```



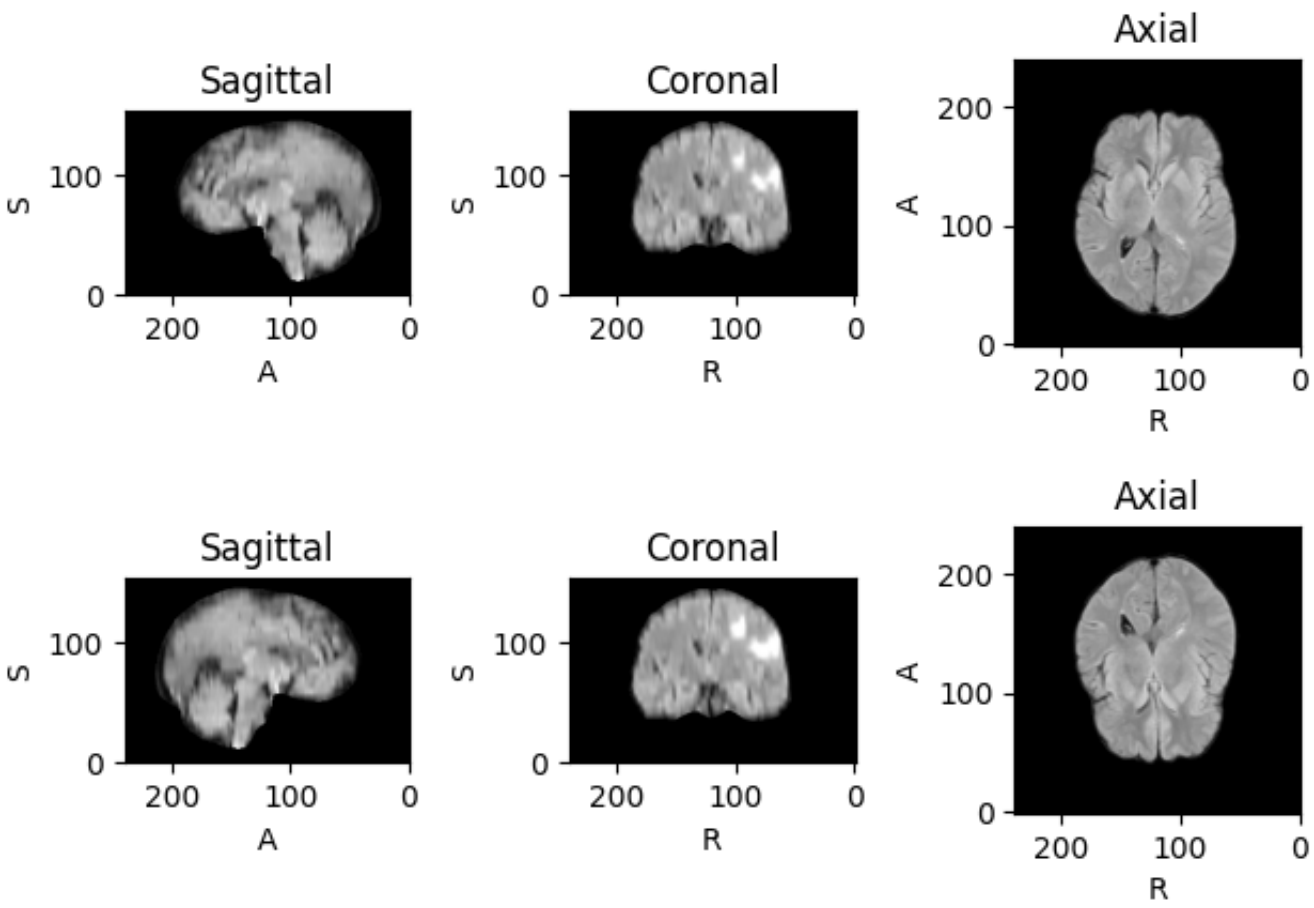

```
MRI Shape: (1, 240, 240, 155)
```

```
# Loading and Viewing Corresponding Segmentation Maps

seg_path = '/content/BraTS-SSA-Samples/BraTS-SSA-00049-000/BraTS-SSA-00049-000-s

seg_sample = tio.ScalarImage(seg_path)
seg_sample.plot()
print("MRI Shape:",seg_sample.shape)
```

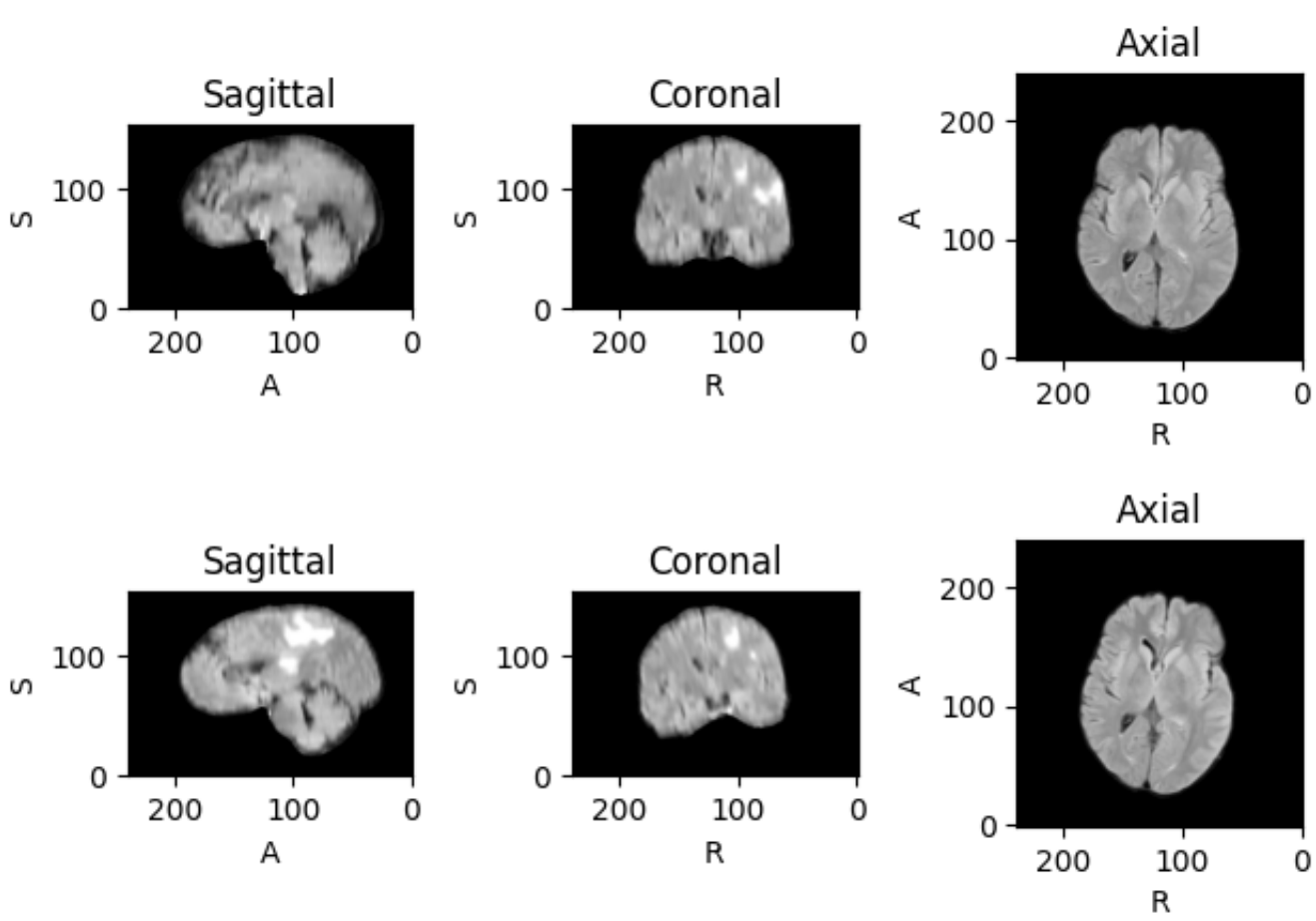



```
MRI Shape: (1, 240, 240, 155)
```

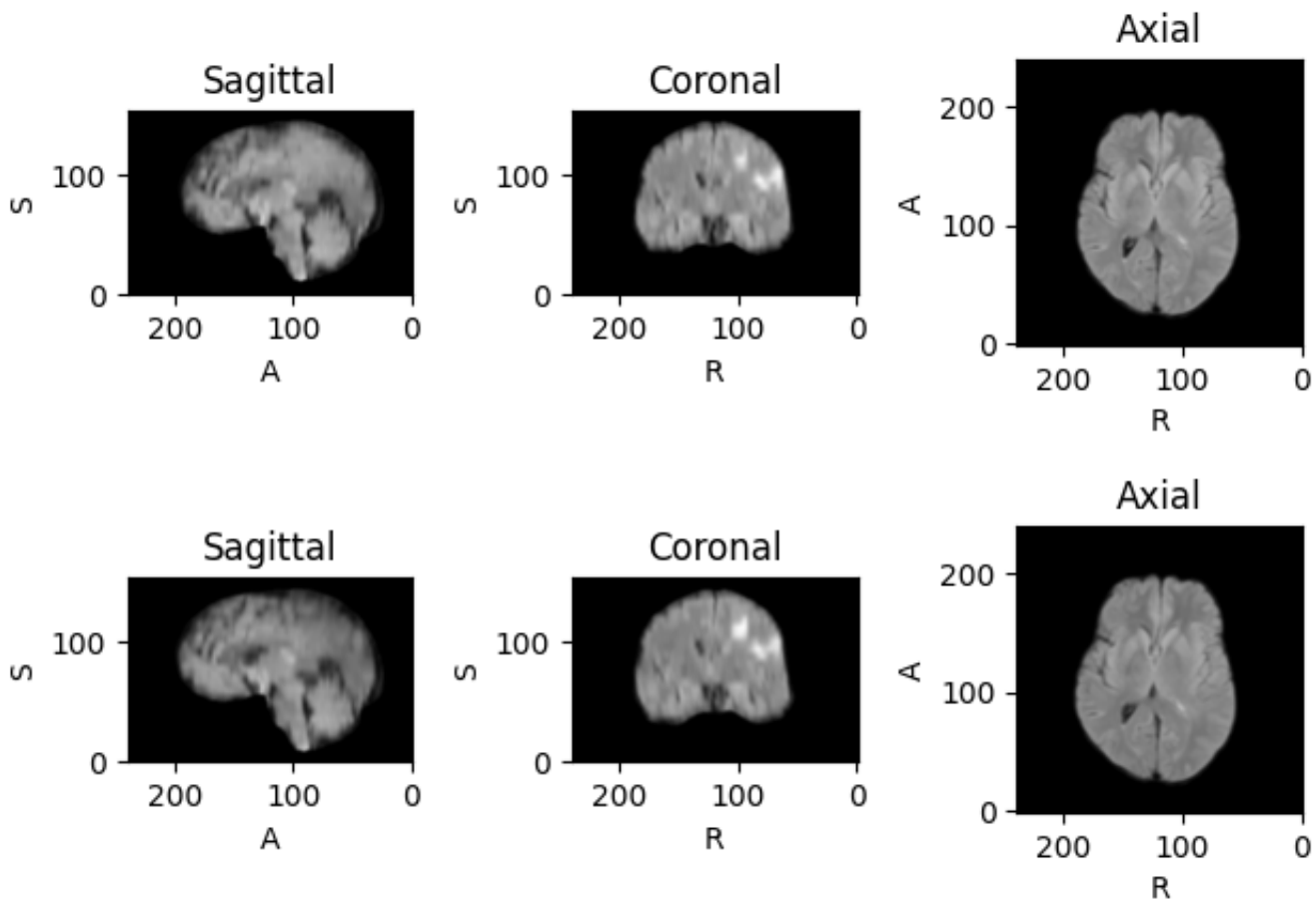
```
# Applying Random Flip to MRI

random_flip = tio.RandomFlip(p=0.9, axes=(0,1,2))
mri_flipped = random_flip(mri_sample)

mri_sample.plot()
mri_flipped.plot()
```



```
# Applying Random Affine to MRI

random_affine = tio.RandomAffine(p=0.9)
mri_affine = random_affine(mri_sample)

mri_sample.plot()
mri_affine.plot()
```



```
# Applying Random Elastic Transformations to MRI

random_elastic = tio.RandomElasticDeformation(p=1)
mri_elastic = random_elastic(mri_sample)

mri_sample.plot()
mri_elastic.plot()
```



Custom Dataset & DataLoader

```
# imports and all ..

import os
from glob import glob

import torchio
from torchio.data import SubjectsLoader, SubjectsDataset
```

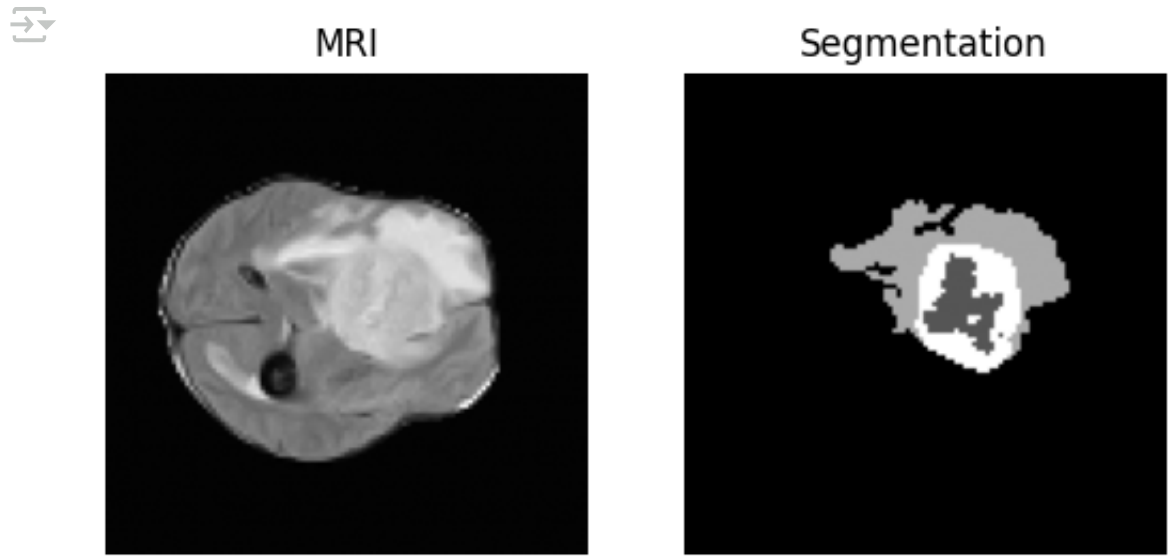
```
# Custom Dataset - 🧠

class SegmentationDataset(SubjectsDataset):
    def __init__(self, root_dir: str, transform=None):
        self.root_dir = root_dir
        self.transform = transform

        # Recursively Finding MRI and Segmentation Label Paths with Glob
        self.t2f_paths = sorted(glob(os.path.join(root_dir, "**", "*t2f*.nii"),
        self.t1c_paths = sorted(glob(os.path.join(root_dir, "**", "*t1c*.nii"),
        self.seg_paths = sorted(glob(os.path.join(root_dir, "**", "*seg*.nii"),

        self.subjects = [
            tio.Subject(
                mri=tio.ScalarImage([t1c_path, t2f_path]),
                label=tio.LabelMap(seg_path)
            )
            for t2f_path, t1c_path, seg_path in zip(self.t2f_paths, self.t1c_pat

        super().__init__(self.subjects, transform=transform)
```

+ Code + Text

```
# Preprocessing Transforms:

preprocessing_transforms = tio.Compose(
    [
        tio.RescaleIntensity(out_min_max=(0, 99.5), include = ['mri']),
        tio.ZNormalization(include = ['mri']),
        tio.Resample((2.2, 2.2, 2.0), include = ['mri', 'label']),
        # tio.CropOrPad((96, 96, 64), include = ['mri', 'label']),
        # tio.OneHot(num_classes = 4, include = ['label']),
    ]
)

# Augmentation Transforms:

augmentation_transforms = tio.Compose(
    [
        tio.RandomFlip(axes=(0,1,2), p=0.5, include = ['mri', 'label']),
        tio.RandomAffine(p=0.5, include = ['mri', 'label']),
        tio.RandomElasticDeformation(p=0.5, include = ['mri', 'label']),
        # tio.RandomNoise(p = 0.3, include = ['mri']),
    ]
)

# All the Transforms

data_transforms = tio.Compose(
    [
        preprocessing_transforms,
        augmentation_transforms
    ]
)


# data transforms, loading

dataset = SegmentationDataset('/content/BraTS-SSA-Samples/', transform=data_tra

dataloader = SubjectsLoader(dataset, batch_size=4, shuffle=True)

batch = next(iter(dataloader))
mri_data = batch['mri'][tio.DATA]
seg_data = batch['label'][tio.DATA]

print(mri_data.shape, seg_data.shape)
```

```
torch.Size([4, 2, 109, 109, 77]) torch.Size([4, 1, 109, 109, 77])
```

```
plt.subplot(1,2,1)
plt.imshow(mri_data[0,1,...,38], cmap = 'gray')
plt.title('MRI')
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(seg_data[0,0,...,38], cmap = 'gray')
plt.title('Segmentation')
plt.axis('off')
plt.show()
```



Start coding or generate with AI.