MedImages.jl

GPU-Accelerated, Differentiable Medical Image Processing in Julia

A unified framework for NIfTI, DICOM, and HDF5 medical imaging workflows

1 Introduction

The Challenge Concept 1

Medical imaging software suffers from fundamental fragmentation:

- **Format proliferation**: NIfTI, DICOM, HDF5, and proprietary formats each require separate tooling
- **Spatial metadata complexity**: Origin, spacing, and direction cosines are handled inconsistently across libraries
- **CPU-only legacy**: Most medical imaging tools predate GPU computing and lack hardware acceleration
- **Non-differentiable operations**: Traditional image processing breaks gradient flow for deep learning integration

Researchers waste time on data wrangling instead of algorithm development.

The Solution Concept 2

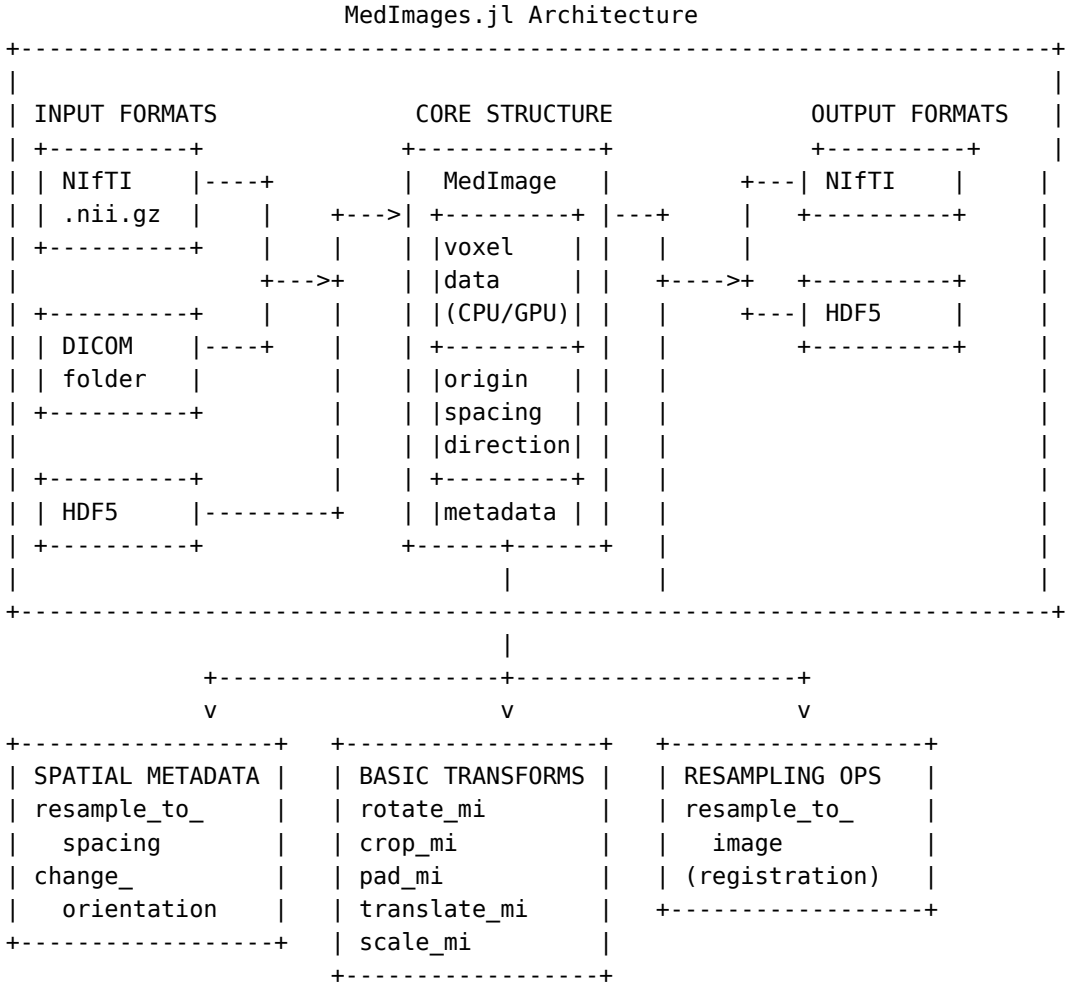
MedImages.jl provides a unified approach to medical image processing:

- **Single data structure**: The `MedImage` struct encapsulates voxel data with complete spatial metadata
- **Format agnostic**: Transparent I/O for NIfTI, DICOM folders, and HDF5 files
- **GPU acceleration**: CUDA support via KernelAbstractions.jl with zero code changes
- **Full differentiability**: Zygote and Enzyme compatible for end-to-end gradient computation
- **Spatial correctness**: All transforms preserve origin, spacing, and orientation metadata

2 Core Architecture

Architecture Overview

Architecture 3



MedImage Data Structure

Architecture 4

The `MedImage` struct is the central data type:

Field	Type	Description
`voxel_data`	`Array`	Multidimensional voxel array (CPU or GPU)
`origin`	`Tuple{F64,F64,F64}`	Physical origin coordinates in mm
`spacing`	`Tuple{F64,F64,F64}`	Voxel dimensions in mm
`direction`	`NTuple{9,F64}`	Direction cosines (3x3 matrix flattened)
`image_type`	`Image_type`	Modality: CT, MRI, or PET
`image_subtype`	`Image_subtype`	Specific acquisition type
`current_device`	`current_device_enum`	CPU or GPU backend indicator
`metadata`	`Dict`	Extensible metadata storage

Type Enumerations — Implementation 5

Image Type Enum

`CT`	Computed Tomography
`MRI`	Magnetic Resonance Imaging
`PET`	Positron Emission Tomography

Image Subtype Enum

`CT_STANDARD`	Standard CT acquisition
`MRI_T1`	T1-weighted MRI
`MRI_T2`	T2-weighted MRI
`MRI_FLAIR`	Fluid-attenuated inversion recovery
`PET_FDG`	FDG-PET metabolic imaging

Device Enum

`CPU_en`	Standard CPU arrays
`GPU_en`	CUDA GPU arrays via CUDA.jl

3 File I/O

File I/O Operations — Implementation 6

Loading Images

```
using MedImages

# Load NIfTI file
ct = load_image("scan.nii.gz", "CT")

# Load DICOM folder
mri = load_image("dicom_folder/", "MRI")

# Access spatial metadata
println("Shape: ", size(ct.voxel_data))
println("Origin: ", ct.origin)
println("Spacing: ", ct.spacing)
println("Direction: ", ct.direction)
```

Saving Images

```
# Export to NIfTI
create_nii_from_medimage(ct, "output.nii.gz")

# Save to HDF5
save_to_hdf5(ct, "output.h5")
```

4 Spatial Concepts

Spatial Coordinate System Concept 7

Medical images exist in physical space, not just voxel indices. MedImages.jl maintains three key spatial properties:

Origin: The physical coordinates (x, y, z) of the first voxel corner in millimeters. This anchors the image in world space.

Spacing: The physical dimensions of each voxel in millimeters. Determines the resolution and scale of the image.

Direction Cosines: A 3x3 rotation matrix (stored as 9 values) defining the orientation of voxel axes relative to world axes. This encodes patient orientation.

All transforms in MedImages.jl automatically update these properties to maintain spatial correctness.

Orientation Codes Architecture 8

Standard orientation codes define anatomical coordinate systems:

Code	Meaning	Use Case
`ORIENTATION_RAS`	Right-Anterior-Superior	Neuroimaging standard (FreeSurfer, FSL)
`ORIENTATION_LPS`	Left-Posterior-Superior	DICOM and ITK standard
`ORIENTATION_RAI`	Right-Anterior-Inferior	Alternative neuroimaging
`ORIENTATION_LPI`	Left-Posterior-Inferior	Alternative DICOM
`ORIENTATION_RPS`	Right-Posterior-Superior	Specialized applications
`ORIENTATION_LAS`	Left-Anterior-Superior	Specialized applications
`ORIENTATION_RPI`	Right-Posterior-Inferior	Specialized applications
`ORIENTATION_LAI`	Left-Anterior-Inferior	Specialized applications

The first letter indicates left-right axis, second indicates anterior-posterior, third indicates superior-inferior.

5 Transformations

Basic Transformations

Implementation 9

All transforms accept an interpolation method and preserve spatial metadata:

```
using MedImages

ct = load_image("scan.nii.gz", "CT")

# Rotate 90 degrees around z-axis
rotated = rotate_mi(ct, :z, 90.0, Linear_en)

# Crop to region of interest (start indices, size)
cropped = crop_mi(ct, (50, 50, 20), (100, 100, 60), Linear_en)

# Pad with zeros (beginning padding, end padding, fill value)
padded = pad_mi(ct, (10, 10, 5), (10, 10, 5), 0.0, Linear_en)

# Translate origin by offset along axis
translated = translate_mi(ct, 25.0, :x, Linear_en)

# Scale by factor (affects spacing)
scaled = scale_mi(ct, 2.0, Linear_en)
```

Interpolation Methods

Architecture 10

Choose interpolation based on your use case:

Method	Speed	Quality	Use Case
`Nearest_neighbour_en`	Fast	Discrete	Segmentation masks, label maps
`Linear_en`	Medium	Good	General CT/MRI processing
`B_spline_en`	Slow	Smooth	High-quality visualization, publication figures

Critical: Always use `Nearest_neighbour_en` for segmentation masks to preserve label integrity. Linear and B-spline interpolation will create invalid intermediate values.

Resampling Operations

Implementation 11

Change Resolution

```
# Resample to isotropic 1mm spacing
isotropic = resample_to_spacing(ct, (1.0, 1.0, 1.0), Linear_en)

# Resample to specific spacing
resampled = resample_to_spacing(ct, (0.5, 0.5, 2.0), Linear_en)
```

Change Orientation

```
# Convert to neuroimaging standard
ras = change_orientation(ct, ORIENTATION_RAS)

# Convert to DICOM standard
lps = change_orientation(ct, ORIENTATION_LPS)
```

Cross-Modal Registration Example 12

Align images from different modalities to a common space:

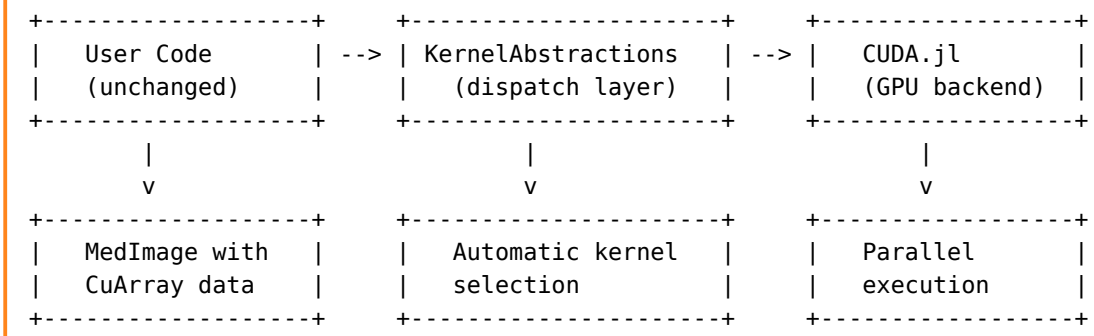
```
# Load CT and PET images
ct = load_image("ct.nii.gz", "CT")
pet = load_image("pet.nii.gz", "PET")

# Resample PET to match CT geometry
# This aligns origin, spacing, and orientation
aligned_pet = resample_to_image(ct, pet, Linear_en)

# Now both images have identical:
# - Voxel grid dimensions
# - Origin coordinates
# - Spacing values
# - Direction cosines

# Save aligned result
create_nii_from_medimage(aligned_pet, "pet_aligned_to_ct.nii.gz")
```

This operation is essential for multi-modal analysis, fusion imaging, and preparing training data for deep learning.

6 GPU Acceleration

GPU Backend Architecture 13

MedImages.jl uses KernelAbstractions.jl for hardware-agnostic GPU kernels:



The same transform functions work on both CPU and GPU data. Backend selection is automatic based on array type.

GPU Usage

Example 14

```
using MedImages
using CUDA

# Load image on CPU
ct = load_image("scan.nii.gz", "CT")

# Transfer to GPU
gpu_ct = deepcopy(ct)
gpu_ct.voxel_data = CuArray(Float32.(ct.voxel_data))
gpu_ct.current_device = GPU_en

# All operations now run on GPU
resampled = resample_to_spacing(gpu_ct, (2.0, 2.0, 2.0), Linear_en)
rotated = rotate_mi(resampled, :z, 45.0, Linear_en)

# Transfer back to CPU for saving
cpu_result = deepcopy(rotated)
cpu_result.voxel_data = Array(rotated.voxel_data)
cpu_result.current_device = CPU_en

create_nii_from_medimage(cpu_result, "output.nii.gz")
```

Differentiability

Concept 15

MedImages.jl supports automatic differentiation through two mechanisms:

Zygote.jl: Reverse-mode AD via ChainRulesCore rrules. Works with standard Julia arrays and integrates with Flux.jl neural networks.

Enzyme.jl: High-performance AD that works with GPU kernels. Required for differentiating through KernelAbstractions code on CUDA.

All resampling and interpolation operations have defined gradients, enabling:

- Spatial transformer networks
- Differentiable rendering
- Image registration with gradient descent
- End-to-end learning with geometric augmentation

```julia
using MedImages
using Zygote

# Define a loss function using MedImages operations
function spatial_loss(voxel_data)
    im = MedImage(
        voxel_data = voxel_data,
        origin = (0.0, 0.0, 0.0),
        spacing = (1.0, 1.0, 1.0),
        direction = (1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0),
        image_type = CT,
        image_subtype = CT_STANDARD,
        current_device = CPU_en,
        metadata = Dict()
    )

    # Differentiable resampling
    resampled = resample_to_spacing(im, (2.0, 2.0, 2.0), Linear_en)

    # Compute scalar loss
    return sum(resampled.voxel_data .^ 2)
end

# Compute gradients
data = rand(Float32, 64, 64, 64)
grads = Zygote.gradient(spatial_loss, data)

# grads[1] contains dL/d(voxel_data)
```

7 Complete Workflow

Complete Pipeline — Example 17

End-to-end example: load, process, and save a medical image:

```
using MedImages

# 1. Load input image
ct = load_image("input_ct.nii.gz", "CT")
println("Original shape: ", size(ct.voxel_data))
println("Original spacing: ", ct.spacing)

# 2. Resample to isotropic 1mm resolution
isotropic = resample_to_spacing(ct, (1.0, 1.0, 1.0), Linear_en)
println("Isotropic shape: ", size(isotropic.voxel_data))

# 3. Reorient to standard neuroimaging convention
ras = change_orientation(isotropic, ORIENTATION_RAS)

# 4. Crop to brain region (example coordinates)
cropped = crop_mi(ras, (50, 30, 20), (160, 200, 180), Linear_en)

# 5. Apply rotation for alignment
aligned = rotate_mi(cropped, :z, 5.0, Linear_en)

# 6. Save processed result
create_nii_from_medimage(aligned, "processed_ct.nii.gz")
println("Processing complete")
```

API Quick Reference

Function	Description
`load_image(path, type)`	Load NIfTI or DICOM files
`create_nii_from_medimage(im, path)`	Export to NIfTI format
`save_to_hdf5(im, path)`	Export to HDF5 format
`resample_to_spacing(im, spacing, interp)`	Change voxel resolution
`change_orientation(im, code)`	Reorient to standard
`resample_to_image(fixed, moving, interp)`	Align moving to fixed geometry
`rotate_mi(im, axis, angle, interp)`	3D rotation around axis
`crop_mi(im, start, size, interp)`	Crop with origin adjustment
`pad_mi(im, beg, end, val, interp)`	Pad with origin adjustment
`translate_mi(im, offset, axis, interp)`	Translate origin
`scale_mi(im, factor, interp)`	Scale image

Interpolation Constants

- `Nearest_neighbour_en` - Discrete values, fast
- `Linear_en` - Smooth interpolation, balanced
- `B_spline_en` - High quality, slow

Orientation Constants

- `ORIENTATION_RAS` - Neuroimaging standard
- `ORIENTATION_LPS` - DICOM standard
- `ORIENTATION_RAI`, `ORIENTATION_LPI`, `ORIENTATION_RPS`, `ORIENTATION_LAS`, `ORIENTATION_RPI`, `ORIENTATION_LAI`