Representing and Communicating AI Model Results in Standard DICOM Format Using the Python Programming Language

Christopher P. Bridge [1], Sean W. Doyle [1], Andriy Y. Fedorov [2], Steven Pieper [3, 4], Erik Ziegler [4], James Petts [4], David A. Clunie [5], Gordon J. Harris [4, 6], Katherine P. Andriole[1, 2], Jochen K. Lennerz [8], Markus D. Herrmann [8]

1 MGH & BWH Center for Clinical Data Science, Boston, MA, USA

2 Department of Radiology, Brigham and Women's Hospital/Harvard Medical School, Boston, MA, USA

3 Isomics, Inc., Cambridge, MA, USA

4 Open Health Imaging Foundation, Boston, MA

5 PixelMed Publishing, LLC., Bangor, PA, USA

6 Department of Radiology, Massachusetts General Hospital/Harvard Medical School, Boston, MA, USA

8 Department of Pathology, Massachusetts General Hospital/Harvard Medical School, Boston, MA, USA

Motivation

- Integrating machine learning (AI) models into clinical workflows requires interoperability with existing imaging systems
- Interoperability requires adherence to standards
- For medical imaging, the dominant standard is DICOM
- The existing DICOM standard has methods for representing many forms of ML model output
- However, working with DICOM can be challenging for AI developers:
    - Many have limited familiarity with DICOM
    - The breadth and complexity of the standard may make it seem inaccessible
    - There has been a lack of DICOM tools that are interoperable with standard ML tooling (the Python programming language, numpy, tensorflow and pytorch)

Agenda

- Review the parts of the DICOM standard that may be appropriate for encoding and communicating AI model results
  - Structured Reports
  - Segmentations
  - Secondary Captures
- Demonstrate, with examples, how to encode and communicate AI model output in these formats using our fully free and open-source (MIT license) *highdicom* and *dicomweb-client* python packages
- Demonstrate how this enables interoperability with existing image storage, communication and display systems
- Target audience: ML developers and software engineers

Why Use DICOM for AI Model Results?

- DICOM objects can be *communicated* and *stored* within existing enterprise imaging systems alongside the images themselves
- Results can be *queried/retrieved* along with original imaging study
- Existing viewers may be able to *display* results stored in DICOM format in an environment familiar to radiologists
- Provides standard fields to enable the traceability required for medical care
- The model for imaging/patient/study metadata is harmonized with with the original images.
  - This additionally allows metadata to cross-reference to other DICOM objects using native DICOM identifiers (e.g. UIDs)

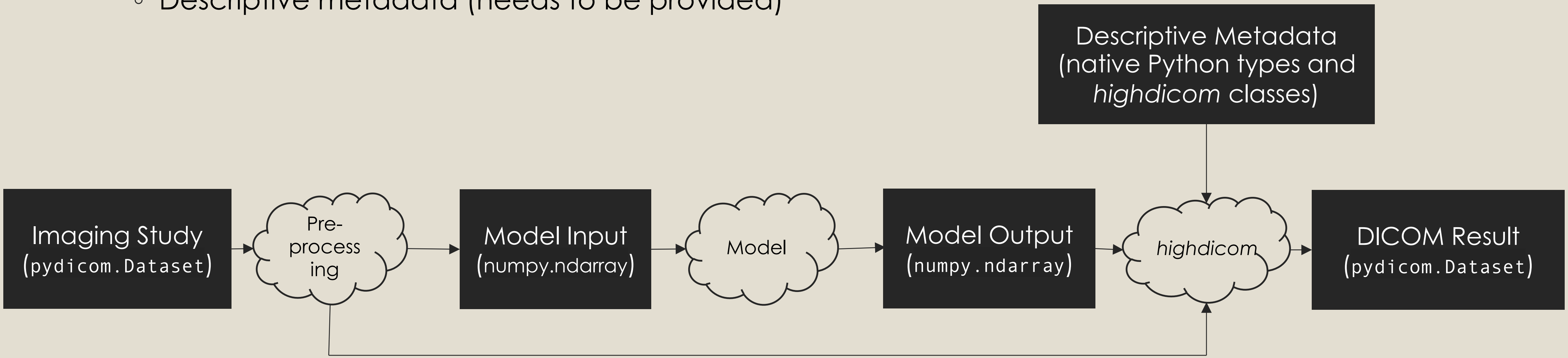
Overview of DICOM IODs for AI Results

- DICOM IODs (Information Object Definitions) are different "classes" of DICOM objects
- We recommend the following IODs for different types of AI model output:

AI Model Output Type	Recommended DICOM IOD
Classification	Structured Report
Object Detection	Structured Report
Segmentation	Segmentation (BINARY type)
Probabilistic Segmentation or Heatmap	Segmentation (FRACTIONAL type)
Other Visual Results	Secondary Capture

Overview of *highdicom*

- Python package providing high-level object-oriented way to create and interface with DICOM objects
- Creates DICOM objects from:
  - Model results (`numpy.ndarray`)
  - Study/patient metadata (intelligently automatically copied from source images)
  - Descriptive metadata (needs to be provided)



highdicom links

- Code on GitHub: https://github.com/mghcomputationalpathology/highdicom
- Documentation on Read the Docs: https://highdicom.readthedocs.io
- Distribution package on PyPI: https://pypi.org/project/highdicom

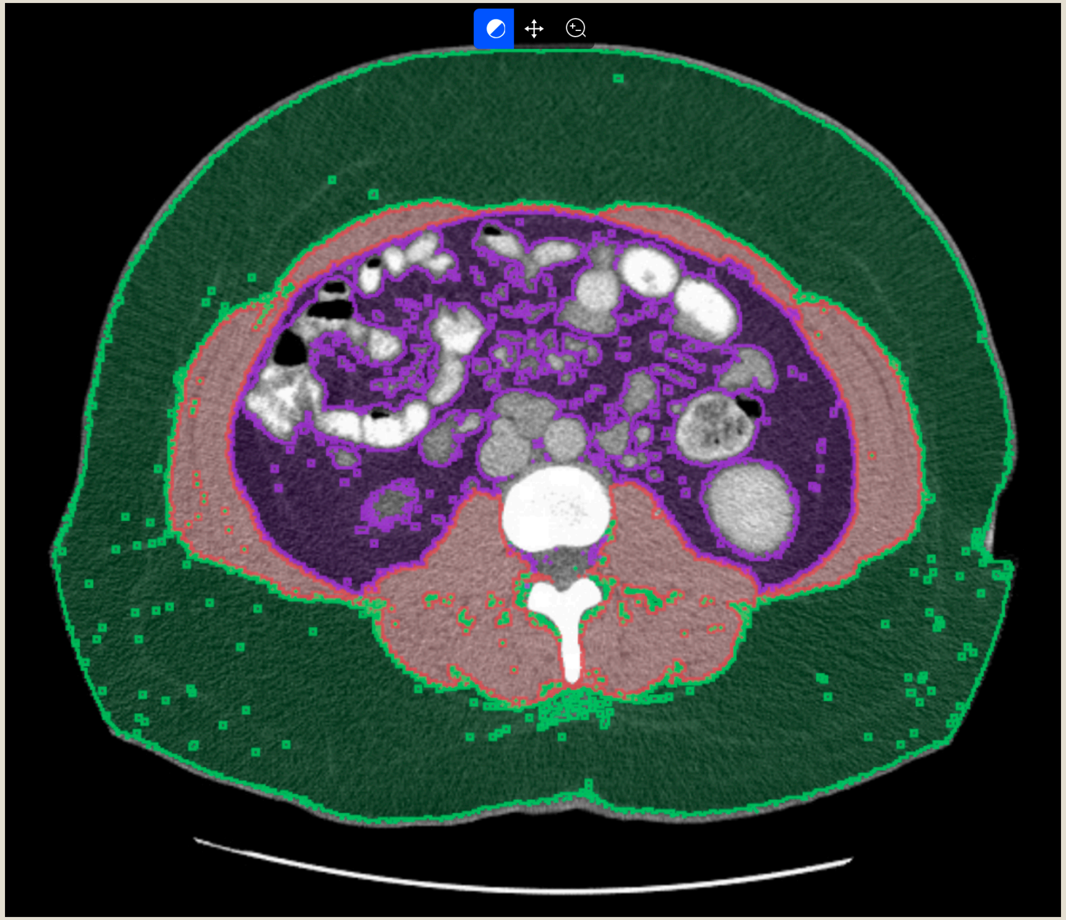
- `$ pip install highdicom`

Classes/Methods Provided By *highdicom*

- Object-oriented interface for construction of DICOM objects o Abstract base class for Service-Object Pair (SOP) Class:
  - `class SOPClass(pydicom.Dataset)`
  - Implementation of SOP Classes for Structured Report (SR) modality (highdicom.sr package):
    - `class EnhancedSR(SOPClass)`
    - `class ComprehensiveSR(SOPClass)`
    - `class Comprehensive3DSR(SOPClass)`
  - Implementation of SOP Classes for Segmentation (SEG) modality (highdicom.seg package):
    - `class Segmentation(SOPClass)`
  - Implementation of SOP Classes for Secondary Capture (highdicom.sc package):
    - `class SCImage(SOPClass)`
- Utility functions for facilitating access of DICOM object content
  - Filtering content of a Structured Report document:
    - `def find_content_items(dataset: pydicom.Dataset, ...) -> List[pydicom.Dataset]`
  - Iterating over segments of a Segmentation image:
    - `def iter_segments(dataset: pydicom.Dataset) -> Generator[numpy.ndarray]`

DICOM Segmentation

- Pixelwise categorization of images into regions of interest stored as raster graphics:
  - Binary, mutually-exclusive multiclass or non-mutually exclusive multiclass
  - Discrete (BINARY type) or probabilistic (FRACTIONAL type)
- Created from segmentation mask as a numpy array
- Accompanied by metadata describing the meaning of each segment



DICOM segmentation containing muscle, visceral fat, and subcutaneous fat segments displayed over the original abdominal CT scan

Segmentation - Ingredients

- pixel_array – The segmentation mask as a numpy.ndarray
- AlgorithmIdentificationSequence - Description of the algorithm used to create the segmentation
- SegmentDescription – Description of the meaning of each segment

Segmentation Example

1. Create descriptions of algorithm and segments

```
from pydicom.sr.codedict import codes
from highdicom.content import (
    AlgorithmIdentificationSequence,
)
from highdicom.seg.content import SegmentDescription
from highdicom.seg.enum import SegmentAlgorithmTypeValues

# Describe the segmentation algorithm used by the model
algorithm = AlgorithmIdentificationSequence(
    name='RSNA2020 Radiology Image Segmentation Example',
    family=codes.cid7162.ArtificialIntelligence,
    version='v0.1.0'
)

# Describe the predicted segment that represents the ROI
segment_description = SegmentDescription(
    segment_number=1,
    segment_label='ROI #1',
    segmented_property_category=codes.cid7150.Tissue,
    segmented_property_type=codes.cid7166.Bone,
    algorithm_type=SegmentAlgorithmTypeValues.AUTOMATIC,
    algorithm_identification=algorithm
)
```

2. Create segmentation

```
from highdicom.seg.enum import SegmentationTypeValues
from highdicom.seg.sop import Segmentation
from highdicom.uid import UID

# Construct Segmentation image instance
segmentation = Segmentation(
    source_images=[image], # type: List[pydicom.Dataset]
    pixel_array=mask, # type: numpy.ndarray
    segmentation_type=SegmentationTypeValues.BINARY,
    segment_descriptions=[segment_description],
    series_instance_uid=UID(),
    series_number=2,
    sop_instance_uid=UID(),
    instance_number=1,
    manufacturer='MGH Radiology',
    manufacturer_model_name='AI Demo',
    software_versions='v1',
    device_serial_number='Device X.Y.Z.'
)

segmentation.save_as('filename.dcm')
```

DICOM Structured Reports (SRs)

- Structured Reports allow for encoding various clinical findings derived from images as structured text/data:
  - Classification results: e.g. existence of clinical findings (referencing existing coding ontologies)
  - Quantification of findings: e.g. volume, severity score
  - Localization results stored as vector graphics (points, bounding boxes, polygons)
- Stored in a hierarchical structure of findings (content tree)
- Various templates are available for defined use cases

Structured Report Example

- In the following code example, we will create a Comprehensive3D SR to describe the area of vertebral foramen in the cervico-thoracic spine derived from a CT image

Example Comprehensive 3D Structured Report Content Tree (Simplified)

- **MeasurementReport**: *describes the measurement/finding*
  - **Observation Context**
    - **Observer Context**: *describes the person or device making the observations*
  - **PlanarROIMeasurementsAndQualitativeEvaluations**: *describes measurements within a defined planar region of interest*
    - **ImageRegion3D**: *a polygon describing the region of interest in the image*
    - **FindingSite**: *description of the anatomical location of region of interest*
    - **Measurement**: *the measurement itself*
      - **Value**
      - **Unit**

Structured Report Example

1. Import relevant classes

```python
import numpy as np
from pydicom.uid import generate_uid
from pydicom.filereader import dcmread
from pydicom.sr.codedict import codes
from highdicom.sr.content import (
    FindingSite,
    ImageRegion3D,
)
from highdicom.sr.enum import GraphicTypeValues3D
from highdicom.sr.sop import Comprehensive3DSR
from highdicom.sr.templates import (
    DeviceObserverIdentifyingAttributes,
    Measurement,
    MeasurementProperties,
    MeasurementReport,
    ObservationContext,
    ObserverContext,
    PersonObserverIdentifyingAttributes,
    PlanarROIMeasurementsAndQualitativeEvaluations,
    TrackingIdentifier,
)
from highdicom.sr.value_types import CodedConcept
```

2. Describe the observing device

```python
# Path to single-frame CT image instance stored as PS3.10 file
image_file = Path('/path/to/image/file')

# Read CT Image data set from PS3.10 files on disk
image_dataset = dcmread(str(image_file))

# Describe the context of reported observations: the person that reported
# the observations and the device that was used to make the observations
observer_person_context = ObserverContext(
    observer_type=codes.DCM.Person,
    observer_identifying_attributes=PersonObserverIdentifyingAttributes(name
='Foo'))
observer_device_context = ObserverContext(
    observer_type=codes.DCM.Device,
    observer_identifying_attributes=DeviceObserverIdentifyingAttributes(
        uid=generate_uid()
    )
)
observation_context = ObservationContext(
    observer_person_context=observer_person_context,
    observer_device_context=observer_device_context,
)
```

Structured Report Example (cont.)

3. Describe the region of interest

```
# Describe the image region for which observations were made
# (in physical space based on the frame of reference)
referenced_region = ImageRegion3D(
    graphic_type=GraphicTypeValues3D.POLYGON,
    graphic_data=np.array([
        (165.0, 200.0, 134.0),
        (170.0, 200.0, 134.0),
        (170.0, 220.0, 134.0),
        (165.0, 220.0, 134.0),
        (165.0, 200.0, 134.0),
    ]),
    frame_of_reference_uid=image_dataset.FrameOfReferenceUID
)


# Describe the anatomic site at which observations were made
finding_sites = [
    FindingSite(
        anatomic_location=codes.SCT.CervicoThoracicSpine,
        topographical_modifier=codes.SCT.VertebralForamen
    ),
]
```

4. Describe the measurement

```
# Describe the imaging measurements for the image region defined above
measurements = [Measurement(
    name=codes.SCT.AreaOfDefinedRegion,
    tracking_identifier=TrackingIdentifier(uid=generate_uid()),
    value=1.7,
    unit=codes.UCUM.SquareMillimeter,
    properties=MeasurementProperties(
        normality=CodedConcept(
            value="17621005",
            meaning="Normal",
            scheme_designator="SCT"
        ),
        level_of_significance=codes.SCT.NotSignificant
    )
)]
imaging_measurements = [PlanarROIMeasurementsAndQualitativeEvaluations(
    tracking_identifier=TrackingIdentifier(
        uid=generate_uid(),
        identifier='Planar ROI Measurements'
    ),
    referenced_region=referenced_region,
    finding_type=codes.SCT.SpinalCord,
    measurements=measurements,
    finding_sites=finding_sites
)]
```

Structured Report Example (cont.)

5. Create the structured report

```
# Create the report content
measurement_report = MeasurementReport(
    observation_context=observation_context,
    procedure_reported=codes.LN.CTUnspecifiedBodyRegion,
    imaging_measurements=imaging_measurements
)


# Create the Structured Report instance
sr_dataset = Comprehensive3DSR(
    evidence=[image_dataset],
    content=measurement_report[0],
    series_number=1,
    series_instance_uid=generate_uid(),
    sop_instance_uid=generate_uid(),
    instance_number=1,
    manufacturer='Manufacturer'
)

sr_dataset.save_as(‘filename.dcm’)
```

DICOM Secondary Capture

- General way to store raster imaging data other than original acquisitions
  - For example images with "burnt-in" graphics layered on top
- Widely supported by viewers
- Created from numpy array of pixels
- Recommended only if SR/SEG are not possible/appropriate
- More specialized IODs should be preferred

Communicating with Imaging Systems via DICOM-Web

- The *dicomweb-client* python package implements a client to communicate over the DICOMweb RESTful API to:
  - Store DICOM objects (STOW-RS), such as AI results in DICOM format
  - Retrieve DICOM objects (WADO-RS), such as model input data
  - Query for studies/series/instances based on metadata (QIDO-RS)
- Can therefore interoperate with most existing enterprise/research imaging systems
- It is also interoperable with both *pydicom* and *highdicom* classes
- Documentation: https://dicomweb-client.readthedocs.io
- Github: https://github.com/MGHComputationalPathology/dicomweb-client
- `pip install dicomweb-client`

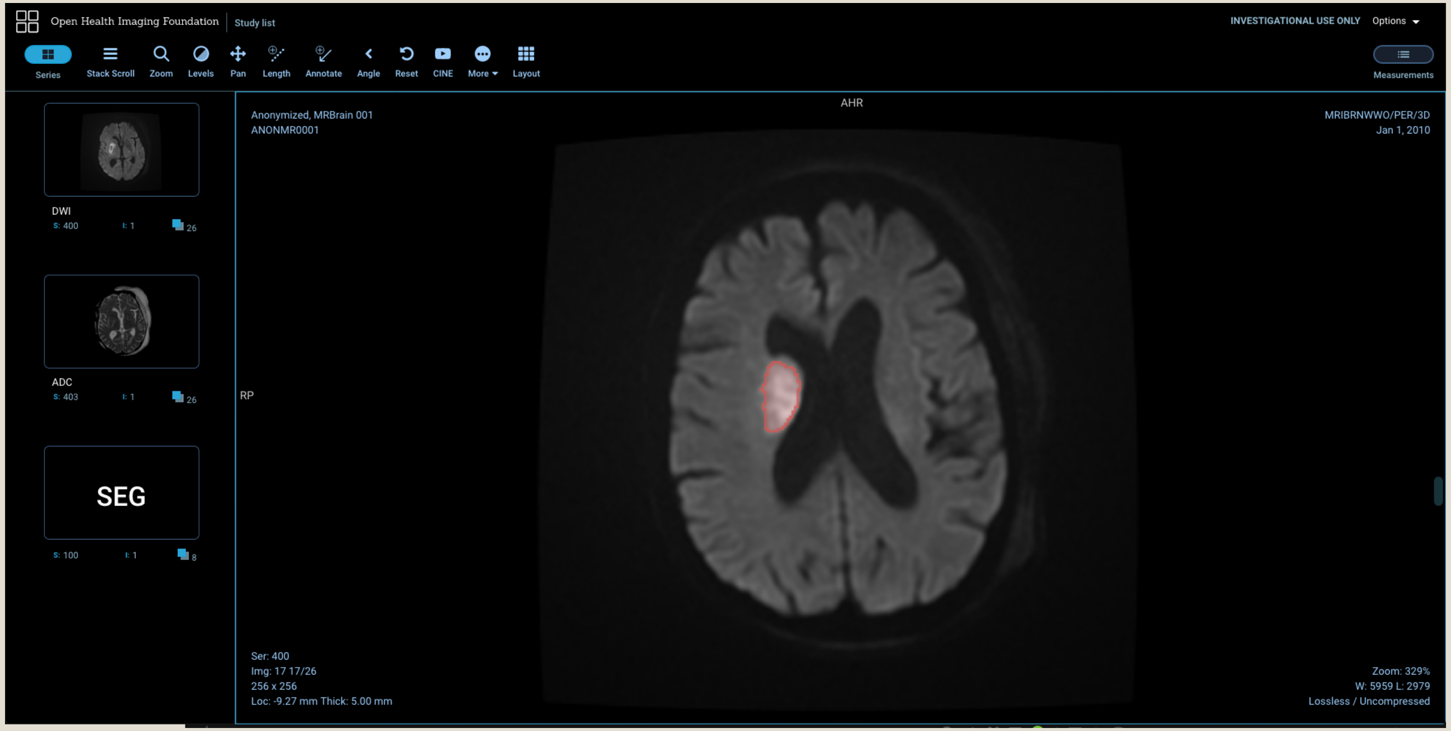
DICOMweb Example

```
from dicomweb_client.api import DICOMwebClient


# Create a client object to communicate to the DICOMweb server
client = DICOMwebClient(url="https://mydicomwebserver.com")


# Pull down a known imaging study for processing
# Returns a list of pydicom datasets
instances = client.retrieve_series(
    study_instance_uid='1.2.826.0.1.3680043.8.1055.1.20111103111148288.98361414.79379639',
    series_instance_uid='1.2.826.0.1.3680043.8.1055.1.20111103111208937.49685336.24517034'
)


# Preprocess datasets, run AI model, encode results in segmentation object
segmentation = ...


# Store the segmentation result
client.store_instances([segmentation])
```

Workflow Example

Full Python-based model integration workflow example:

- Image read using *pydicom*
- Segmentation via *tensorflow* model
- Stored as DICOM segmentation using *highdicom*
- Communicated via DICOMweb to open-source DICOM server Orthanc with *dicomweb-client* package
- *Rendered* by the open-source web-based OHIF viewer



Summary

- Encoding AI model results in DICOM format can ease integration of model into existing clinical workflows
- We have described the parts of the DICOM standard appropriate for containing AI model results or different types
- High-level open-source python packages are available for creating and communicating DICOM objects with interoperability with numpy/pytorch/tensorflow
- This enables model developers to integrate with existing systems from within a fully Python-based environment

- Contact:

  - Christopher Bridge (cbridge@partners.org)
  - Markus Herrmann (mdherrmann@mgh.harvard.edu)

Acknowledgements

- MGH & BWH Center for Clinical Data Science (CCDS): https://www.ccds.io
- Alliance for Digital Pathology: https://digitalpathologyalliance.org
- Quantitative Image Informatics for Cancer Research (QIICR): http://qiicr.org
- Radiomics: https://www.radiomics.io
- National Alliance for Medical Image Computing (NA-MIC): https://www.na-mic.org
- Open Health Imaging Foundation (OHIF): http://ohif.org
- Imaging Data Commons (IDC): https://datascience.cancer.gov/data-commons
- *highdicom* and *dicomweb-client* contributors