WHY YOUR REGISTRATION LIBRARY SHOULD OUTPUT AN ITK TRANSFORM

LINKS

- Itk wrapper in icon_registration (lovingly commented)
  - https://github.com/uncbiag/ICON/blob/master/src/icon_registration/itk_wrapper.py
- Notebook to follow along:
  - https://colab.research.google.com/drive/1NAFFGCD2hh84kfkCQpNhSC4wK4v4MVIq?usp=sharing

THE NIGHTMARE SCENARIO

- My registration code takes as input two (1, 1, 128, 128, 128) tensors I_fixed and I_moving, and outputs a (1, 3, 128, 128, 128) tensor Φ_inv.
- I_fixed ~= torch.nn.functional.grid_sample(I_moving, Φ_inv). Success!
- Later, an MD-PHD gives me two DICOM images to register. One is 130x130x110 with spacing (1, 8, .7333) and the other is 220 x 220 x 100 with spacing (.5, .5, .9). Both have orientation [[0, 1, 0], [1, 0, 0], [0, 0, -1]]
- I say, "easy," and resample the images to 128x128x128, register them, and email the MD-PHD a pickle of a shape (1, 3, 128, 128, 128) tensor.
- They stare at me like I'm a cat presenting its owner with a dead mouse.

REGISTERED!

TWO PROBLEMS

- Downstream medical application code needs to correctly handle image spacing, image orientation, and unusual image origins correctly – in particular, landmarks, volume and size measurements, and mesh data are always in physical coordinates
- If we try to manually handle this metadata by, for each downstream task, manually scaling, transposing, and shifting deformation fields, we will make mistakes and spend a lot of time.

  This is notably different from segmentations, where it is trivial to just rescale the output labelmap, copy over the metadata, and call it a day

TWO PRIORITIES

- 1: No matter what, record transforms in a way that allows transforming physical coordinates
- 2: Maybe we can all record transforms in the same way?

- We need a lingua franca datatype for transforms like itk.Image is for images
  - That understands orientation, spacing, origin
  - That has a correct way to compose transforms, warp images, and transport points

PROPOSAL

- Write a wrapper for each of our registration libraries that can take input images as itk.Images with associated metadata, register them, and return an itk.Transform.

ITK.TRANSFORM

- C++ class with various subclasses, wrapped in python
- Composition, warping points, warping images, computing jacobians, inversion of some types of transforms all implemented in physical coordinates, respecting image metadata
- Serializable
- Integrated with ITK ecosystem – Slicer!

ICON_REGISTRATION NOW FOLLOWS THIS CONVENTION

- pip install icon_registration

- Pretrained model available for knee mri and brain mri
- Following code from icon_registration
  - https://github.com/uncbiag/ICON/blob/master/src/icon_registration/itk_wrapper.py

```
[ ] import icon_registration.itk_wrapper as itk_wrapper
    import icon_registration.pretrained_models as pretrained_models
    import itk
    import matplotlib.pyplot as plt
```

```
[ ] model = pretrained_models.OAI_knees_registration_model()

    #Feel free to experiment with different images here
    image_A = itk.imread("9487462_20081003_SAG_3D_DESS_RIGHT_11495603_image.nii.gz")
    image_B = itk.imread("9225063_20090413_SAG_3D_DESS_RIGHT_12784112_image.nii.gz")

    phi_AB, phi_BA = itk_wrapper.register_pair(model, image_A, image_B)
```

```
Downloading pretrained model (1.2 GB)
```

```
[ ] interpolator = itk.LinearInterpolateImageFunction.New(image_A)
    warped_image_A = itk.resample_image_filter(image_A,
        transform=phi_AB,
        interpolator=interpolator,
        size=itk.size(image_B),
        output_spacing=itk.spacing(image_B),
        output_direction=image_B.GetDirection(),
        output_origin=image_B.GetOrigin()
    )
    plt.imshow(image_A[80])
    plt.show()
    plt.imshow(image_B[80])
    plt.show()

    plt.imshow(warped_image_A[80])
    plt.show()

    plt.imshow(itk.checker_board_image_filter(warped_image_A, image_B)[80])
    plt.show()
```

USER CODE: LOW RISK OF BUGS

CRIMINALLY UNDERRATED ITK FUNCTIONS

These are super useful, whether or not you follow this proposal:

- Image.TransformPhysicalPointToContinuousIndex
- Image.TransformContinuousIndexToPhysicalPoint

With this proposal, this would become equally useful:

- Transform.TransformPoint

COMPLEXITY MOVED TO LIBRARY CODE

```
def register_pair(model, image_A, image_B):

    assert( isinstance(image_A, itk.Image))
    assert( isinstance(image_B, itk.Image))
    icon_registration.network_wrappers.adjust_batch_size(model, 1)
    model.to(config.device)

    A_npy = np.array(image_A)
    B_npy = np.array(image_B)
    A_trch = torch.Tensor(A_npy).to(config.device)[None, None]
    B_trch = torch.Tensor(B_npy).to(config.device)[None, None]

    shape = model.identityMap.shape

    A_resized = F.interpolate(A_trch, size=shape[2:], mode="trilinear", align_corners=False)
    B_resized = F.interpolate(B_trch, size=shape[2:], mode="trilinear", align_corners=False)

    with torch.no_grad():
        x = model(A_resized, B_resized)

    phi_AB = model.phi_AB(model.identityMap)[0].cpu()
    phi_BA = model.phi_BA(model.identityMap)[0].cpu()

    return (
        create_itk_transform(phi_AB, model.identityMap, image_A, image_B),
        create_itk_transform(phi_BA, model.identityMap, image_B, image_A)
    )
```

COMPLEXITY MOVED TO LIBRARY CODE

```
def create_itk_transform(phi, ident, image_A, image_B):

    disp = phi - ident[0].cpu()

    network_shape_list = list(ident.shape[2:])

    dimension = len(network_shape_list)

    scale = torch.Tensor(network_shape_list)
    for _ in network_shape_list:
        scale = scale[:, None]
    disp *= scale
    tr = itk.DisplacementFieldTransform[(itk.D, dimension)].New()

    disp_itk_format  = disp.double().numpy()[list(reversed(range(dimension)))].transpose(list(range(1, dimension + 1)) + [0])

    itk_disp_field = array_to_vector_image(disp_itk_format)

    tr.SetDisplacementField(itk_disp_field)

    to_aligned = resampling_transform(image_A, list(reversed(network_shape_list)))

    from_aligned = resampling_transform(image_B, list(reversed(network_shape_list))).GetInverseTransform()

    phi_AB_itk = itk.CompositeTransform[itk.D, dimension].New()

    phi_AB_itk.PrependTransform(from_aligned)
    phi_AB_itk.PrependTransform(tr)
    phi_AB_itk.PrependTransform(to_aligned)

    return phi_AB_itk
```

PREFER TO REPRESENT TRANSFORMS AS COMPOSITIONS

- Instead of attempting to post facto modify a deformation field to respect orientation and spacing, it is better to represent a transform as a composition of
  - An affine transform to go from physical coordinates to the scaled coordinates used by our registration algorithm
  - A deformable transform with the deformation field from our neural network
    - This deformable transform must be represented as a vector image with consistent, but not necessarily physical, metadata
  - An affine transform to go from our nonphysical internal coordinates to physical coordinates.
- As a result, it is critical that compositions are a first class citizen of our transform datatype

WORTHWHILE TO WRITE TRICKY LIBRARY CODE INSTEAD OF ASKING USERS TO WRITE TRICKY APPLICATION CODE!

SERIALIZATION AND DESERIALIZATION

```
itk.transformwrite([phi_AB], "phi_AB_.hdf5")
```

```
phi_AB_from_disk = itk.transformread("phi_AB_.hdf5")[0]
```

SHOULD WE USE ITK.TRANSFORM?

- Pros:
  - APIs for warping an image with metadata, composing transforms, and warping physical points are unambiguous
  - Can switch between deformable transform, affine transform, spline transform without changing application code
  - Integrated closely with itk images, which are the most common way of representing medical images with metadata
  - Does not have to integrate into deep learning code
  - Medical people are (begrudgingly) familiar with ITK
  - Correctness is easy to verify
  - Slicer Compatibility
  - Serialization!
- Cons:
  - Python APIs for warping an image, composing transforms, and warping physical points are clunky [itk type system]
  - Has to be a wrapper: cannot integrate into deep learning code / differentiate through

WHO'S ON BOARD?

- Full compatibility
  - 3D Slicer
  - ITK Registration
  - icon_registration
  - SimpleITK
- In Flight
  - ITK_elastix
  - ANTs: internal representation is itk.Transform?
  - Perhaps MONAI?

Speak up and correct me on this slide!