CT Artery Segmentation

Assign	
Status	In Progress
Priority	Medium
Date Created	@January 26, 2022 12:05 PM
Due Date	

Steps

Installation and Environment

```
conda create -n art_seg python=3.10

conda install -c conda-forge tensorflow-gpu=2.7
pip install nibabel
pip install notebook
pip install matplotlib
pip install opencv-python
pip install patchify
pip install -U scikit-learn
pip install torchio
```

Data

Gathered data

17 volumes for **training**

2 for **testing**

Preprocessing

Need to figure out how to make size same for all volumes

- Trying keras example method first
  - resize and normalize
  - Check if volumes and masks are consistent
    - Applying preprocessing
    - Need to visualize results now (write visualize function)
      - I think resulting `masks` and `slices` are not consistent

```
# Testing resize consistency (between vol and mask)  with one image
vol_path = "./data/train/volumes/1.nii.gz"
vol = process_scan(vol_path)
mask_path = "./data/train/masks/1.nii.gz"
mask = process_scan(mask_path)

vol_slice = vol[:,431,:]
vol_slice = cv2.rotate(vol_slice, cv2.ROTATE_90_COUNTERCLOCKWISE)
vol_slice = cv2.flip(vol_slice, 1)
plt.imshow(vol_slice, cmap='gray')

mask_slice = mask[:,:,255]
mask_slice = cv2.rotate(mask_slice, cv2.ROTATE_90_COUNTERCLOCKWISE)
mask_slice = cv2.flip(mask_slice, 1)
plt.imshow(mask_slice, cmap='gray')
```

      - Thee seem consistent now

- Only problem might be the one with `88 slices` along one axis?
  - I checked that image (resampled from **88** → **512**)
    - It also looks consistent enough
- I checked same slice and its mask in **Slicer** and then also in my `script`. Both looked same.

I can proceed to next step now, but have to `visualize` a number of slices i.e. **coronal** instead of **sagittal**.

```
*insert plot funciton here
```

Label maps

Need to check label-maps → should be two

- Color table file shows the following labels

```
0 Background 0 0 0 0
1 temporal_artery 128 174 128 255
2 facial_artery 216 101 79 255
```

- In slicer, if you bring the cursor to points, you can read the label values as 0,1 and 2.

Mask resize

Label masks values are changed from 0,1 and 2 to floating values after resize function i.e. `skimage.transform.resize` and `scipy.ndimage.zoom`

Changed `ndimagezoom` to `skimage.transform.resize` because **ndimage** is not giving floating values. While **skimage resize** gives same values 0,1 and 2.

Made `resize` function simpler compared to keras example. Also, removed `normalization` for masks.

Custom colormap

Created custom colormap function for mask visualization.

```
def set_mask_cmap(img):
    color_dict = {'Background': 'black','temporal_artery': 'green','facial_artery': 'red'}
    if 1 in img:
        custom_cmap = matplotlib.colors.ListedColormap([color_dict['Background'],
                                                        color_dict['temporal_artery']])
    elif 2 in img:
        custom_cmap = matplotlib.colors.ListedColormap([color_dict['Background'],
                                                        color_dict['facial_artery']])
    elif 1 in img and 2 in img:
        custom_cmap = matplotlib.colors.ListedColormap([color_dict['Background'],
                                                        color_dict['temporal_artery'],
                                                        color_dict['facial_artery']])
    else:
        custom_cmap = matplotlib.colors.ListedColormap(color_dict['Background'])
    return custom_cmap
```

Define the model

```
# Building an encoder-decoder 3D UNet architecture
```

```
# Defining convolutional block
def conv_block(_input, num_filters):
    x = layers.Conv3D(num_filters, 3, padding='same')(_input)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    x = layers.Conv3D(num_filters, 3 ,padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    return x

# Encoder block
def encoder_block(_input, num_filters):
    x = conv_block(_input, num_filters)
    p = layers.MaxPooling3D((2,2,2))(x)
    return x, p

# Decoder block
def decoder_block(_input, skip_features, num_filters):
    x = layers.Conv3DTranspose(num_filters, (2,2,2), strides=2, padding='same')(_input)
    x = layers.Concatenate()([x, skip_features])
    x = conv_block(x, num_filters)
    return x

# Build UNet using blocks
def build_unet(input_shape, num_classes):
    inputs = layers.Input(input_shape)

    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    b1 = conv_block(p4, 1024)

    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    if num_classes == 1:
        activation = 'sigmoid'  # Binary case
    else:
        activation = 'softmax'  # Categorical case

    outputs = layers.Conv3D(num_classes, 1, padding='same', activation=activation)(d4)
    print(activation)

    model = Model(inputs, outputs, name='UNet-3D')
    return model
```

Dataloader

Taking help from this link → Defining custom data generator in keras.

> We are doing that because keras generator only understands `jpeg`, `tiff` and `png` files. Not `nii` or `npy`.

Also this link and

https://www.youtube.com/watch?v=ScdCQgLtnis&t=829s

```
"""
Custom data generator to work with nifti files.
We will load data as well as apply preprocessing here.
"""

# Load and process volume
def load_volumes(vol_dir, vol_list):
    volumes = []
    for i, vol_name in enumerate(vol_list):
        if (vol_name.split('.')[1] == 'nii'):
            volume = process_scan(vol_dir+vol_name, mode='scan')
```

```
            volumes.append(volume)
    volumes = np.array(volumes)

    return volumes

# Load and process mask
def load_masks(mask_dir, mask_list):
    masks = []
    for i, mask_name in enumerate(mask_list):
        if (mask_name.split('.')[1] == 'nii'):
            mask = process_scan(mask_dir+mask_name, mode='mask')
            mask = np.round(mask)
            mask[mask==3] = 2  # Set extra label to 2 (facial)
            masks.append(mask)
    masks = np.array(masks)

    return masks

def ImageLoader(vol_dir, mask_dir):
    vol_list = os.listdir(vol_dir)
    mask_list = os.listdir(mask_dir)
    print(mask_list)
    L = len(vol_list)

    while True:  # Keras needs it to be infinite
        batch_start = 0
        batch_end = batch_size

        while batch_start < L:
            limit = (min(batch_end, L))

            X = load_volumes(vol_dir, vol_list[batch_start:limit])
            Y = load_masks(mask_dir, mask_list[batch_start:limit])

            yield (X,Y)

            batch_start += batch_size
            batch_end += batch_size
```

Issue

Unique values is return 0, 1, 2, 3 instead of 0, 1, 2 → `6.nii.gz`

- set extra class 3 to 2

```
# Load and process mask
def load_masks(mask_dir, mask_list):
    masks = []
    for i, mask_name in enumerate(mask_list):
        if (mask_name.split('.')[1] == 'nii'):
            mask = process_scan(mask_dir+mask_name, mode='mask')
            mask = np.round(mask)
            mask[mask==3] = 2
            used_classes = np.unique(mask)
            print(used_classes)
            masks.append(mask)
    masks = np.array(masks)

    return masks
```

Floating values → `3, 4.nii.gz`

- Fixed with **np.round()**

```
# Load and process mask
def load_masks(mask_dir, mask_list):
    masks = []
    for i, mask_name in enumerate(mask_list):
        if (mask_name.split('.')[1] == 'nii'):
            mask = process_scan(mask_dir+mask_name, mode='mask')
            mask = np.round(mask)
            mask[mask==3] = 2
            used_classes = np.unique(mask)
            print(used_classes)
            masks.append(mask)
    masks = np.array(masks)

    return masks
```

Validation data

I will have sufficient data after augmentation, so need to think how to make `valdiation split`.

- Should I separate data manually in separate folders?
  - and make a separate `datagen` for it?
- Should I keep `30%` data for validation during batch batch?
  - Then that means I have to perform validation in each batch `iteration`?
    - But in reality we perform validation after each `epoch`, not with each iteration
- Think!

Augmentation

- https://github.com/fepegar/torchio
  - https://colab.research.google.com/github/fepegar/torchio-notebooks/blob/main/notebooks/TorchIO_tutorial.ipynb
  - https://torchio.readthedocs.io/transforms/transforms.html
  - https://torchio.readthedocs.io/transforms/preprocessing.html#onehot

Automatically decided which `augmentation` should be applied to **Label Maps** and which not.

Testing. Need to apply same transformations to `volume` and `masks`.

Can follow the above `colab` notebook to augment both `volume` and `mask` together. Then can convert the ouput `dict` to a list and save them separately.

To `save` → https://github.com/fepegar/torchio/discussions/731

Checking if `volume` and `mask` have same actions applied to them and output is ok.

- It is good for `Flip` augmentation
- Test with a `compose()` set of augmentations
  - Works fine

```
transform = tio.Compose([
    tio.RandomFlip(axes=(0,1), flip_probability=1),
    tio.RandomMotion(p=0.5),
    tio.RandomBiasField(p=0.3),
    tio.RandomNoise(p=0.5),
    tio.RandomFlip(),
    tio.OneOf({
        tio.RandomAffine(
            scales=(0.9, 1.2),
            degrees=15
        ): 0.8,
        tio.RandomElasticDeformation(): 0.2,
    }),
])
```

Now need to finalize the `augmentation` that I am going to use after reading the docs, and the n make a `loop` for it.

- Current idea → make two dicts with different `augmentation`
  - Use one dict to apply a single transformation with `torchio.OneOf()`

- Use other one to apply a `compose` transformation
- Something like `OneOf(transform_compose, (OneOf(transform_dict))`

First of all, test all the `transformations` separately by saving the result and visualizing in **3D Slicer**.

- To determine correct argument ranges.
- Tested separately and also combined.

check `Znormalization` → how to apply it on training set in pre-processing?

- No, sticking with custom normalization function.

Writing final clean script with loops. Will generate **50** transformations per patient. Following transformation are used.

```
"""
Reference Link: https://torchio.readthedocs.io/transforms/augmentation.html#augmentation
Check the link for more details.
"""

# Define transformations inside dicts
transform_dict = {
    tio.Resample(4),
    tio.RandomMotion(),
    tio.RandomBiasField(),
    tio.RandomNoise(std=(0,400)),
    tio.RandomFlip(axes=(0,1), flip_probability=1),
    tio.RandomAnisotropy(axes=(0,1,2), downsampling=6),
    tio.RandomGhosting(),
    tio.RandomSpike(),
    tio.RandomBlur(std=(0,3)),
    tio.RandomSwap(patch_size=30),
    tio.RandomGamma(log_gamma=(-0.5, 0.5)),
    tio.OneOf({
        tio.RandomAffine(
            scales=(0.9, 1.2),
            degrees=(15, 15)
        ): 0.8,
        tio.RandomElasticDeformation(): 0.2,
    }),
}

# Composed transformation to apply multiple transformations at once
transform_compose = tio.Compose([
    tio.Resample(p=0.3, target=4),
    tio.RandomMotion(p=0.2),
    tio.RandomBiasField(p=0.3),
    tio.RandomNoise(p=0.5, std=(0,400)),
    tio.RandomFlip(axes=(0,1), flip_probability=0.5),
    tio.RandomAnisotropy(p=0.3, axes=(0,1,2), downsampling=6),
    tio.RandomGhosting(p=0.2),
    tio.RandomSpike(p=0.3),
    tio.RandomBlur(p=0.3, std=(0,3)),
    tio.RandomSwap(p=0.3, patch_size=30),
    tio.RandomGamma(p=0.3, log_gamma=(-0.5, 0.5)),
    tio.OneOf(p=0.3, transforms = {
        tio.RandomAffine(
            scales=(0.9, 1.2),
            degrees=(15, 15)
        ): 0.8,
        tio.RandomElasticDeformation(): 0.2,
    }),
])
```

```
# Define the final transform function
# We will apply compose transformation with 20% probability
# And single transformations with 80%

transform = tio.OneOf({transform_compose: 0.2, tio.OneOf(transform_dict): 0.8})
```

Storage issue

I don't have enough storage to store such huge amount of data (around 200 GB).

Two options.

- Use shared network drive
  - But very slow read/write
  - Training will also be slow
- Use external HDD
  - I used my HDD but it got corrupted and doesn't work anymore
  - Will try 980 SSD after I get the adapter case.
- For now I tried network, half process was done but then ran into an error in one file.
- Copied data from network to `980 SSD`, will use that now for training.

Spatial shape error during runtime

```
RuntimeError: More than one value for "spatial_shape" found in subject images:
{'mask': (512, 512, 79), 'vol': (512, 512, 88)}
```

The error was in `5.nii.gz`.

Will test it by running that file separately.

Try adding an exception `if statement` for size mismatch in `mask` and `volume`. Then perform `CropOrPad` within the exception code.

Apparently, the mask has `Axial` size of `79` instead of `88`.

- Plan is to `resize` the `mask` to `88` and then
- No, excluding it from training. Heejin is not sure which ct volume is used for labeling.

Data split

Need to split data into `training` and `validation`. Will think about `test` data later.

I have `generator` which will load data from the device, so I cannot really split by using `index split` or `sklearn test_split`.

Should I do it manually?

Found a really useful library split-folders. It automatically divides data into train, val and test folders for each folder/class.

```
data
|-volumes
|-masks

splitfolders.ratio('data', output='output', seed=1, ratio=(.75, 0.25))
```

Wrote a `data_split` script to divide data into train and val folders. Also verified if the split for `volumes` and `masks` is same and not different.

Patches

Using `patchify` to convert data from `DataLoader` into patches. Need to test it and check consistency of `volumes` and `masks`.

OR

Check without patches??

Let's try without `patches` first and see what happens.

Training

Approach 1 (Keras Unet3D)

Ran into memory issue. Need to train with patches.

I think my `pathching` methodology is wrong. Even though I am diving into patches, I think I am still feeding multiple patches simultaneously that cause memory error. I am feeding `512x64x64x64` instead of `64x64x64x64`. Meaning, feed 64 patches at a time instead of 512.

I need to modify generator with respect to `patches`.

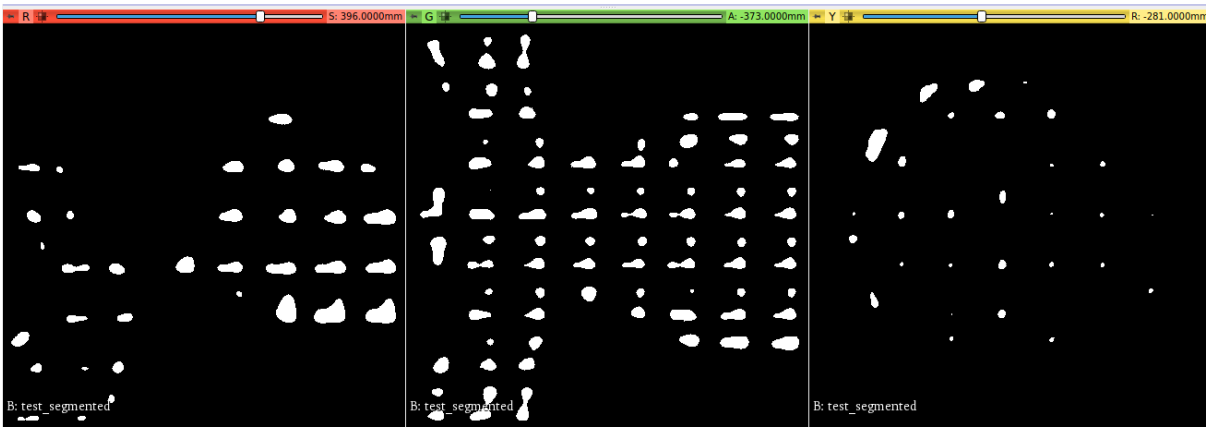
Do I need image-level `batch_size`? I don't think so. I just need to divide the 64 patches from generator into batches.

Training started but seems wrong.

Trained for five epochs. Loss went really down which is not a good sign.

Not even sure if it is using all the patches from a single volume.

Results are bad. Also, only two labels exist.



I think generator is messed up.

Torch IO example (approach 2)

Let's try Torch IO example approach.

RAM and Swap issues for whole day. Tried reducing `barch_size`, `patch_size` etc.

Not working.

The issue is with `num_workers`. Setting it to `1` worked.

Reference → https://stackoverflow.com/questions/68756034/pytorch-problem-my-jupyter-stuck-when-num-workers-0

```
When num_workers is greater than 0, PyTorch uses multiple processes for data loading.

Jupyter notebooks have known issues with multiprocessing.
```

I am also able to use large `batch_size` and `patch_size`.

ERROR

I got the error during loss calculation.

```
Target tensor size: 2
Output tensor size: 3
```

I figured out the cause of error.

Not all data `masks` have 3 labels. In some masks, one of the arteries is missing (missing class) and hence the label size is 2 instead of 3 which give the error.

```
count_equal = 0
count_diff = 0
for sample in validation_dataset:
    transform = tio.OneHot()
    transformed = transform(sample)
    print(transformed.image.data.shape, transformed.label.data.shape)
    if transformed.label.data.shape[0] == 3:
        print("True")
        count_equal += 1
    else:
        print("False")
        count_diff += 1
```

Using script `label_mismatch_test` to figure out the files/volumes/patients which have this issue.

```
images_dir = os.path.join(dataset_dir, 'train/volumes')
labels_dir = os.path.join(dataset_dir, 'train/masks')
image_paths = sorted(glob.glob(images_dir + '/' + '*.nii.gz'))
label_paths = sorted(glob.glob(labels_dir + '/' + '*.nii.gz'))
assert len(image_paths) == len(label_paths)

training_subjects = []
for (image_path, label_path) in zip(image_paths, label_paths):
    subject = tio.Subject(
        image=tio.ScalarImage(image_path),
        label=tio.LabelMap(label_path),
    )
    training_subjects.append(subject)
training_dataset = tio.SubjectsDataset(training_subjects)
print('Dataset size:', len(training_dataset), 'training subjects')
```

```
# Training data
issue_masks = []

for label_path in label_paths:
    label = tio.LabelMap(label_path)
    unique_labels = np.unique(label.data)

    # Check label sizes and save the issue masks
    if unique_labels.shape != (3,):
        print("Size is not correct.")
        print(label_path.split("/")[-1])
        issue_masks.append(label_path.split("/")[-1])
```

```
# Check which labels are missing

for label_path in glob.glob(validation_labels_dir + '/*.nii.gz'):
#     print(mask)
#     print(mask.split('/')[-1].split('.')[0].split('_')[0])
#     break
    split = label_path.split('/')[-1].split('.')[0].split('_')[0]

    if split == '2':
        print(f"*** Patient {split} ***")
        label = tio.LabelMap(label_path)
        unique_labels = np.unique(label.data)
        print(unique_labels)

    if split == '6':
        print(f"*** Patient {split} ***")
        label = tio.LabelMap(label_path)
        unique_labels = np.unique(label.data)
        print(unique_labels)
```

Seems like:

```
Patient 2 and 6 have missing labels.
Patient 2-> missing facial_artery
Patient 6-> missing nothing, but has extra label
```

Found out that `patient 2` is missing `facial_Artery` while `patient 6` is not missing anything. He just has one extra class which I set equal to class 2 in pre-processing.

The `extra` class corresponds to left facial artery i.e. label 2 and 3 correspond to left and right facial artery. So I can just combine them in one label by something like `mask[mask==3]=2`.

Extra class fix

Online is taking really long time.

```
images_dir = os.path.join(dataset_dir, 'train/volumes')
labels_dir = os.path.join(dataset_dir, 'train/masks')
image_paths = sorted(glob.glob(images_dir + '/' + '*.nii.gz'))
label_paths = sorted(glob.glob(labels_dir + '/' + '*.nii.gz'))
assert len(image_paths) == len(label_paths)

training_subjects_fixed = []
for (image_path, label_path) in zip(image_paths, label_paths):
    label = tio.LabelMap(label_path)
    unique_labels = np.unique(label.data)

    if len(unique_labels) > 3:
        print(label_path)
    label.data[label.data==3] = 2

    subject = tio.Subject(
        image=tio.ScalarImage(image_path),
        label=label
    )

    training_subjects_fixed.append(subject)

# print('Dataset size:', len(training_subjects_fixed), 'training subjects')
```

Need to do it offline. Change the values and save the mask again.

```
# Save all paths
```

```
issue_file_paths = []

for label_path in glob.glob(labels_dir + '/*.nii.gz'):
    split = label_path.split('/')[-1].split('.')[0].split('_')[0]

    if split == '6':
        issue_file_paths.append(label_path)
```

```
for path in issue_file_paths:
    name = path.split("/")[-1]
    label = tio.LabelMap(sample_path)
    label.data[label.data==3] = 2
    label.save(f"./temp/train/{name}")
```

Check:

```
validation_fixed = "./temp/val"
for subject in glob.glob(validation_fixed + "/*.nii.gz"):
    label = tio.LabelMap(subject)
    unique_labels = np.unique(label.data)
    print(unique_labels)
```

Now replace old data with new one for this patient on the disk.

Missing Label Fix (loss function issue)

The issue is with `loss function` I think. It should support multi-class.

It does support multi-class, torch IO example used it.

Then what should I do?

If target is missing a label, it should still be able to calculate loss for me.

I think I figured it out maybe. Loss needs to be **multi-class dice loss** but it is **binary class** I think. The Torch IO example have only 2 output classes so they used `binary loss`.

DICE LOSS is still giving me size mismatch error. I think size really needs to be same for it i.e. of `target` and `output`.

I tried BCE before but there was some size issue. Maybe it was because of the fact that my `targets` were not in right format. It should not be `one hot` I think.

Now trying **BCE** now as mentioned here in `LongTensor` format:

https://discuss.pytorch.org/t/multiclass-segmentation/54065/2?page=3

Spacing Issue

- Ran into issue
  - `ToCanonical()` also gave same error.
  - Maybe I can try `Resample()`.
  - https://github.com/fepegar/torchio/issues/647#issuecomment-913025695

```
RuntimeError: As described above, some images in the subject are not in the same space. You probably can use the transforms ToCanonica
```

- I think this issue occurred because I included `tio.Resample(3)` in the transforms. I thought it was resizing, resolution or something but it is actually related to **physical space** and **output spacing**.

```
subjects = []
for (image_path, label_path) in zip(vol_paths, masks_paths):
    subject = tio.Subject(
        image=tio.ScalarImage(image_path),
        label=tio.LabelMap(label_path),
    )
    subjects.append(subject)
```

```
for subject in subjects:
    print(subject.items())
    print("\n")
    print("\n")
```

- I can try to resample all images to a fixed space of $(0.4, 0.4, 0.5)$.

```
# On all images
transform = tio.Resample(target=(0.4,0.4,0.5))
for subject in subjects:
    output = transform(subject)
    print(output.spacing)
```

- I can try this online during preprocessing. If not then I have to do it offline before the training.
  - So online Resampling is taking very long time. There can be two ways now.
    - I `resample` all the images and then do the training. However, that might not be unnecessary because the volumes with `tio.Resample` are defected ones.
    - Good approach will be to get those volumes and resample only those.
- I am checking resampling values to figure out which volumes to resample. Not sure whether I need to resample all or just a few.

  ```
  # Check if spacing is absolutely equal
  for subject in subjects:
      target = (0.4,0.4,0.5)
      spacing = subject["image"].spacing
      x = True if (spacing == target) else False
      print(x)
      print("\n")
  ```

  - Return False because floating values differ a little.
  - However, training ran which mean small difference does not affect. Significant one does.
  - So I need to check spacing values for whole data and then filter with some range.
    - There are some defected volumes with spacing of (3,3,3) and (4,4,4). So I will just resample those and save.

    ```
    # Apply on training data

    transform = tio.Resample(target=(0.4,0.4,0.5))

    for image_path,label_path in zip(train_vol_paths, train_masks_paths):
        subject = tio.Subject(
            image=tio.ScalarImage(image_path),
            label=tio.LabelMap(label_path),
        )

        if subject['image'].spacing == (3,3,3) or subject['image'].spacing == (4,4,4):
            output = transform(subject)
            name = image_path.split('/')[-1]
    ```

```
        output.image.save(os.path.join(out_train_vol_dir, f'{name}'))
        output.label.save(os.path.join(out_train_mask_dir, f'{name}'))
```

```
# Check spacing in output data
out_train_vol_paths = sorted(glob.glob(out_train_vol_dir + '/*.nii.gz'))
out_train_masks_paths = sorted(glob.glob(out_train_mask_dir + '/*.nii.gz'))

for image_path,label_path in zip(out_train_vol_paths, out_train_masks_paths):
    image=tio.ScalarImage(image_path)
    label=tio.LabelMap(label_path)
    print(image.spacing, label.spacing)
```

Should also try **Dice Loss** without `one hot` I think.

Well I was silly. If I define the `num_classes` in the `one hot` transform function during preprocessing, then the `target` shape will always be C=3 with labels (0,1,2) instead of just (0,1).

So now I have two **configurations**.

1. BCE loss without one-hot targets (LongTensors)
2. Dice Loss with one-hot labels

Training with **Dice Loss** first.

Origin Issue

Ran into this.

```
RuntimeError: More than one value for "origin" found in subject images:
{'image': (-119.80000039935112, 43.19998434185982, 415.0),
 'label': (-117.8000003695488, 45.19998437166214, 415.0)}
```

So I think I need to resample whole data set to same space.

Looped through whole data to `resample` it if it wasn't already.

```
# Apply on training data

transform = tio.Resample(target=(0.4,0.4,0.5))

for image_path,label_path in zip(train_vol_paths, train_masks_paths):
    subject = tio.Subject(
        image=tio.ScalarImage(image_path),
        label=tio.LabelMap(label_path),
    )

    if subject['image'].spacing != (0.4000000059604645, 0.4000000059604645, 0.5):  # To avoid applying on the already resampeld images
        output = transform(subject)
        name = image_path.split('/')[-1]
        output.image.save(os.path.join(out_train_vol_dir, f'{name}'))
        output.label.save(os.path.join(out_train_mask_dir, f'{name}'))
```

The issue still persisted.

Turns out some of my subjects had different `origin` for `image` and `labelafter tio.ToCanonical().`

I ran a script to find out the issue files.

```
 # Compare after Canonical
count = 0
transform = tio.ToCanonical()

for image_path,label_path in zip(validation_vol_paths, validation_masks_paths):
    subject = tio.Subject(
        image=tio.ScalarImage(image_path),
        label=tio.LabelMap(label_path),
    )

    output = transform(subject)
    x = True if output['image'].origin != output['label'].origin else False

    if x == True:
        print(image_path)
        print(label_path)
        count += 1

print(count)
```

Issue files

```
1 in validation data -> 6_42
4 in train data -> 6_21, 6_3, 6_41, 6_46
```

Fix

I used `Slicer3D` to register mask to volume in issue files.

Steps

- Open `mask` and `volume` in slicer.
- Go to Registration → General Registration.
- Select fixed and moving volumes, also the output volume
- Select `Rigid 6 DOF` registration and apply
- Then save the mask

After that I verified the mask and volume origins of issue files with following script.

```
import glob, os
import torchio as tio

filename = '6_3'

vol = '/home/trojan/skia_projects/3D-CT-Artery-Segmentation/temp/{}_vol.nii.gz'.format(filename)
mask = '/home/trojan/skia_projects/3D-CT-Artery-Segmentation/temp/{}_mask.nii.gz'.format(filename)
mask_fixed = '/home/trojan/skia_projects/3D-CT-Artery-Segmentation/temp/{}_mask_fixed.nii.gz'.format(filename)

image = tio.ScalarImage(vol)
label = tio.LabelMap(mask)
label_fixed = tio.LabelMap(mask_fixed)

print(image.origin)
print(label.origin)
print(label_fixed.origin)

transform = tio.ToCanonical()
out_image = transform(image)
out_label = transform(label)
out_label_fixed = transform(label_fixed)

print(out_image.origin)
print(out_label.origin)
print(out_label_fixed.origin)
```

```
DATA_DIR = '/home/trojan/skia_projects/3D-CT-Artery-Segmentation/temp'
volumes_dir = os.path.join(DATA_DIR, 'volumes')
```

```
masks_dir = os.path.join(DATA_DIR, 'masks')
fixed_dir = os.path.join(DATA_DIR, 'fixed')

# Before Canonical
for filename in os.listdir(volumes_dir):
    image = tio.ScalarImage(os.path.join(volumes_dir, filename))
    label = tio.LabelMap(os.path.join(masks_dir, filename))
    label_fixed = tio.LabelMap(os.path.join(fixed_dir, filename))
    print(filename)
    print(image.origin)
    print(label.origin)
    print(label_fixed.origin)
    print('\n')

# After Canonical
transform = tio.ToCanonical()

for filename in os.listdir(volumes_dir):
    image = tio.ScalarImage(os.path.join(volumes_dir, filename))
    label = tio.LabelMap(os.path.join(masks_dir, filename))
    label_fixed = tio.LabelMap(os.path.join(fixed_dir, filename))
    out_image = transform(image)
    out_label = transform(label)
    out_label_fixed = transform(label_fixed)
    print(filename)
    print(out_image.origin)
    print(out_label.origin)
    print(out_label_fixed.origin)
    print('\n')
```

Replaced the `masks` in the dataset.

Got GPU memory error after first validation epoch. Reduced batch size.

The `kernel` died during training epoch.

I think I need to reduce patch size.

Training working with

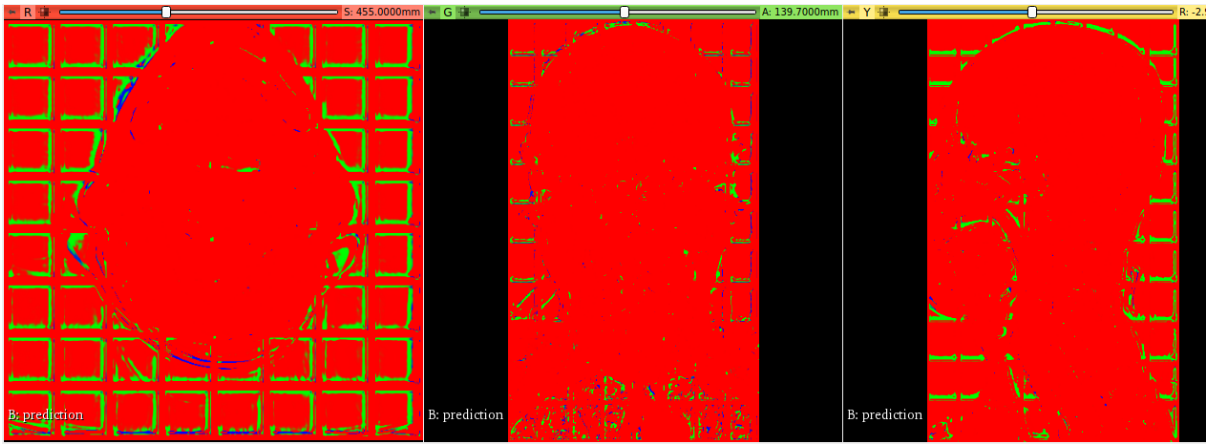
```
Patch size: 64,64,64
Batch size: 4
Patches per sample: 512
```

Results

Dice loss

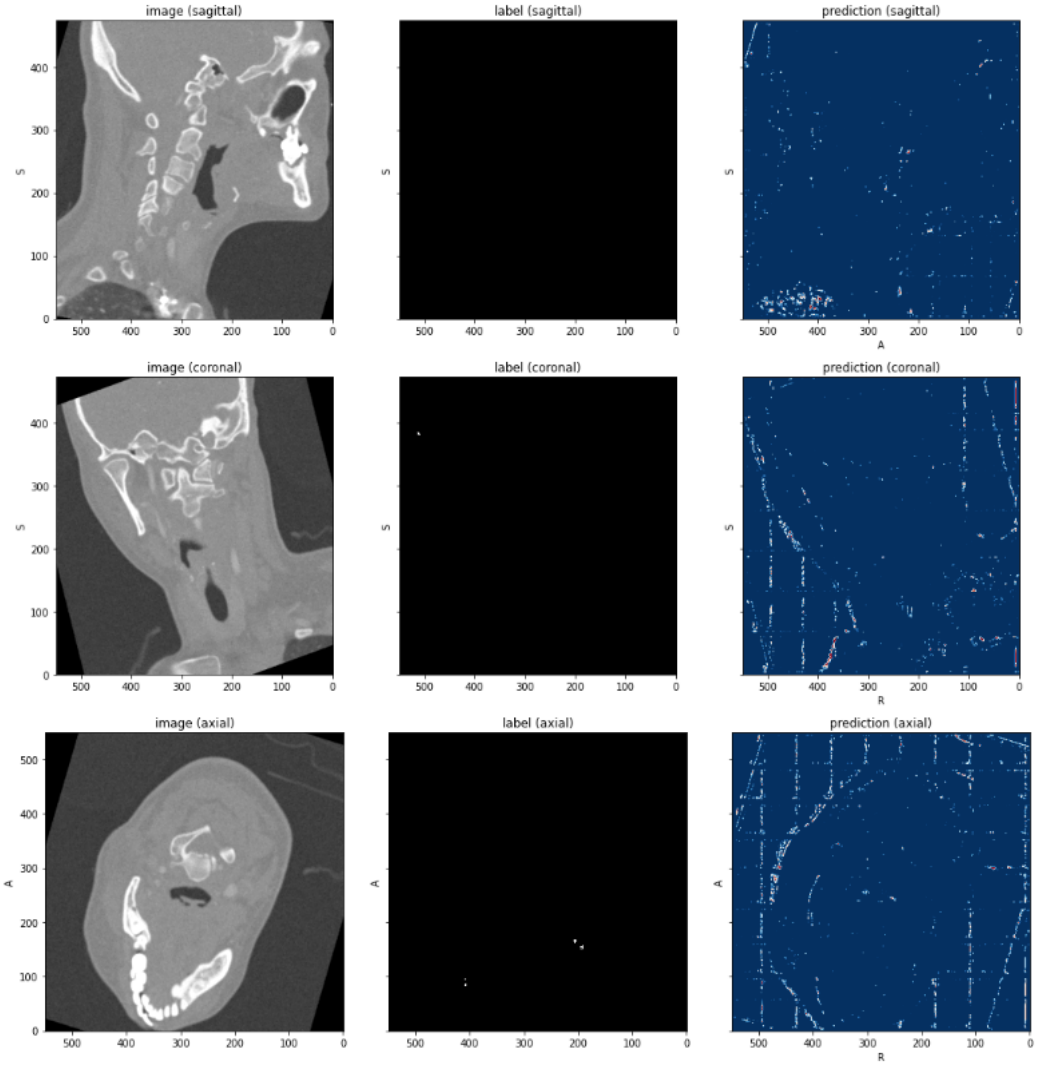
The following loss was used and results were checked.

```
def dice_loss(output, target, epsilon=1e-9):
    p0 = output
    g0 = target
    p1 = 1 - p0
    g1 = 1 - g0
    tp = (p0 * g0).sum(dim=SPATIAL_DIMENSIONS)
    fp = (p0 * g1).sum(dim=SPATIAL_DIMENSIONS)
    fn = (p1 * g0).sum(dim=SPATIAL_DIMENSIONS)
    num = 2 * tp
    denom = 2 * tp + fp + fn + epsilon
    dice_score = num / denom
    return 1 - dice_score
```



As we can see, result are not good.

```
Green -> temporal_artery
Blue -> facial_artery
```



Remarks

I think there can be various reasons for such results.

- Class imbalance
- Less data
- Using all slices (should use only the artery ones maybe?)
  - This can also reduce volume size

Channel-wise dice loss

Currently training with channel-wise dice loss:

```
def dice_loss(output, target):
    smooth = 1.
    loss = 0.
    for c in range(NUM_CLASSES):
        oflat = output[:, c ].contiguous().view(-1)
        tflat = target[:, c].contiguous().view(-1)
        intersection = (oflat * tflat).sum()

        #w = class_weights[c]
        loss += (1 - ((2. * intersection + smooth) /
                          (oflat.sum() + tflat.sum() + smooth)))
    return loss
```

Results still not good. Loss not decreasing.

Focal Loss with 512x512x16 patches

Now training with `focal loss` with `sigmoid` activation.

It will take long time to `train`, so I am gonna implement `focal loss` with `512x512x16` to save time. → Reference

Too slow..

Working with keras example again

I am trying to work with keras example again now.

I am resizing volume to `64x64x512` and then patching it to `64x64x16`.

I need to check if I am yielding and training on all the patches.

Results not good. Also not sure if generator is correct.

Torch IO with `128x128x512`

Patch size → `128x128x32`

Using Resize function now.

Online is taking like forever. Need to do it offline and save.

Training now, not sure if it will work. Probably not.

I am training with `128x128x128` instead and `batch_size` of 4. I am using 128x128x128 * 4 times per patient.

The reason why I am not using `512x512x512` is that then there are a lot of patches and training is extremely slow.

Results

Not good. Loss did not go down either.

Saving and cleaning patches?

Thinking about saving `512x512x16` patches on the disk before training, and then cleaning them (removing unimportant slices).

OR

slice with useful range and ignore rest of the slices.

MONAI (approach 3)

Reference Link

https://www.youtube.com/watch?v=AU4KlXKKnac&list=PLQCkKRar9trODKvJr2B2A3P2m-9Wk_ziF

My Work

Data Preparation

Making Groups

For now we will leave `width` and `height` same i.e. `512x512`

However, we separate the slices into groups with fixed number of slices.

- If all the `patients` have almost same number of slices, then we do not need to do this.
  - We can just crop each patient to a fixed number and drop some of the slices
- However, if the slices are not same and have considerable difference, then to make them ready for training, we need to divide them in equal groups
- Main reason is that we need consistent data for training and also we cannot feed all `512` slices to the network due to memory issues. So we divide them into groups of `64` slices each.

Write here why we are converting to groups of 65???? MUST

Nifti to Dicom

- Reference → https://pycad.co/nifti2dicom/
  - Didn't work
- Tried my code for converting from nibabel (nifti) arrays
  - Results not correct
- 3D Slicer way work but it is manual

- Found a github repo
  - there was some issue with `floating values`
    - I followed this to solve it

```
########################################
castFilter = sitk.CastImageFilter()
castFilter.SetOutputPixelType(sitk.sitkInt16)
```

```
# Convert floating type image (imgSmooth) to int type (imgFiltered)
new_img = castFilter.Execute(new_img)

########################################
```

- Converting the data now

- The code to separate dicom files into groups of 65 slices has some issue
  - It is skipping slices and resulting images are bad
- Turns out my approach with nifti had similar results. So instead of spending so much time in conversion from nifti to dicom, I could have just used nifti files.
  - Trying that again now

```
import os, glob
import nibabel as nib
import numpy as np

from tqdm.notebook import tqdm, tnrange
from tqdm.contrib import tzip

# Do for train then val
in_path = '/media/trojan/evo/3D-CT-Artery-Segmentation/data_splitted/train/volumes'
out_path = '/media/trojan/evo/3D-CT-Artery-Segmentation/data_64_groups/train/images'

for patient in tqdm(glob.glob(in_path + '/*')):
    patient_name = os.path.basename(os.path.normpath(patient)).split('.')[0]
    img = nib.load(patient)
    data = img.get_fdata()
    print(data.shape)
    number_folders = int(data.shape[2] / 64)

    i = 0
    for idx in tnrange(number_folders):
        output_path_name = os.path.join(out_path, patient_name + '_' + str(idx))
        if i+65 > data.shape[2]:
            break
        out_img = nib.Nifti1Image(data[:,:,i:i+65], None)
        nib.save(out_img, os.path.join(output_path_name + '.nii.gz'))
        i+=64
```

Removing empty volumes

Wrote a script to remove the volumes that only contain the `background` class. Such volumes are not useful for training.

```
import os, glob
import numpy as np
import nibabel as nib
from tqdm.contrib import tzip

# Do for train then val
data_dir = '/media/trojan/evo/3D-CT-Artery-Segmentation/temp/data'
images_path = os.path.join(data_dir, 'images')
labels_path = os.path.join(data_dir, 'labels')

list_labels = sorted(glob.glob(labels_path + '/*'))
list_images = sorted(glob.glob(images_path + '/*'))

cnt = 0
for image,label in tzip(list_images,list_labels):
    nifti_file = nib.load(label)
    fdata = nifti_file.get_fdata()
    unique_labels = np.unique(fdata)
    if len(unique_labels) == 1:
        !rm -r {label} && rm -r {image}
        cnt+=1
    #print(f"{label.split('/')[-1]}: {unique_labels}")
```

Before cleaning

```
Training group volumes: 4685
Validation group volumes: 1650
```

After cleaning

```
Training group volumes: 2845
Validation group volumes: 955
```

Installing packages

Install

- Monai
- Pytorch
- Matplotlib

```
conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
pip install monai
pip install matplotlib
```

Data preprocessing

Loading data and applying some transformations. Transformers should be applied in same order. The letter `d` in transforms stand for dictionary. Dictionaries with `keywords` makes it easier to write code and access elements at different parts.

```
def prepare(in_dir, pixdim=(1.5,1.5,1.0), a_min=-400, a_max=400, spatial_size=[128,128,64], cache=False):
    """
    This is the preprocessing function with some of the basic transforms from monai documentation.
    http://monai.io/docs.html
    """

    set_determinism(seed=0)

    # Get and prepare data
    train_images_path = os.path.join(in_dir, 'train/images')
    train_labels_path = os.path.join(in_dir, 'train/labels')
    val_images_path = os.path.join(in_dir, 'val/images')
    val_labels_path = os.path.join(in_dir, 'val/labels')

    train_images_list = sorted(glob.glob(train_images_path + '/*.nii.gz'))
    train_labels_list = sorted(glob.glob(train_labels_path + '/*.nii.gz'))
    val_images_list = sorted(glob.glob(val_images_path + '/*.nii.gz'))
    val_labels_list = sorted(glob.glob(val_labels_path + '/*.nii.gz'))

    train_files = [{'vol': image_name, 'seg': label_name} for image_name,label_name in zip(train_images_list, train_labels_list)]
    val_files = [{'vol': image_name, 'seg': label_name} for image_name,label_name in zip(val_images_list, val_labels_list)]

    # Define transform function
    train_transforms = Compose(
        [
            LoadImaged(keys=['vol', 'seg']),  # Load image before anything else
            AddChanneld(keys=['vol', 'seg']),
            Spacingd(keys=['vol', 'seg'], pixdim=pixdim, mode=('bilinear', 'nearest')),
            Orientationd(keys=['vol', 'seg'], axcodes='RAS'),
            ScaleIntensityRanged(keys=['vol'], a_min=a_min, a_max=a_max, b_min=0.0, b_max=1.0, clip=True),
            CropForegroundd(keys=['vol', 'seg'], source_key='vol'),
            Resized(keys=['vol', 'seg'], spatial_size=spatial_size),
            ToTensord(keys=['vol', 'seg']),
        ]
    )

    val_transforms = Compose(
        [
            LoadImaged(keys=['vol', 'seg']),  # Load image before anything else
            AddChanneld(keys=['vol', 'seg']),
            Spacingd(keys=['vol', 'seg'], pixdim=pixdim, mode=('bilinear', 'nearest')),
            Orientationd(keys=['vol', 'seg'], axcodes='RAS'),
            ScaleIntensityRanged(keys=['vol'], a_min=a_min, a_max=a_max, b_min=0.0, b_max=1.0, clip=True),
            CropForegroundd(keys=['vol', 'seg'], source_key='vol'),  # Deletes background black region
```

```
            Resized(keys=['vol', 'seg'], spatial_size=spatial_size),
            ToTensord(keys=['vol', 'seg']),
        ]
    )

    if cache:
        train_ds = CacheDataset(data=train_files, transform=train_transforms, cache_rate=1.0)
        train_loader = DataLoader(train_ds, batch_size=1)

        val_ds = CacheDataset(data=val_files, transform=val_transforms, cache_rate=1.0)
        val_laoder = DataLoader(val_ds, batch_size=1)

        return train_loader, val_loader

    else:
        train_ds = Dataset(data=train_files, transform=train_transforms)
        train_loader = DataLoader(train_ds, batch_size=1)

        val_ds = Dataset(data=val_files, transform=val_transforms)
        val_laoder = DataLoader(val_ds, batch_size=1)

        return train_loader, val_loader
```

Training

RUN: 1

Trained the model with simple `Dice Loss`.

- The loss went down but then it got stuck after few epoch and stayed the same.

I applied `softmax` activation to the results but couldn't see any thing in the output **label map**/

However, after I applied `sigmoid` activation, output predict some labels but not accurate.

Now I need to check if output has 2 classes or 3. Also need to figure out using softmax.

RUN: 2

Try to follow MONAI example code. Also, try UNETR instead of UNET.

Use `CacheDataset` this time. It will load data into GPU and speed might be faster.

Use `DiceCELoss` with weighting.

So configuration will be as follows:

- Weighted DiceCELoss
- CacheDataset
- Multi-class

Better to make a new separate notebook for this.

Results don't seem too good.

Data Restructuring

I am using too many volumes which is making training really slow. Lets just do 2-3 augmentations per patient. People have used smaller datasets.

Copying original 16 patients data from final data to avoid `origin` and `spacing` issue again.

Augmented each patient `six` times. Total `112` volumes now. Will divide into val and train.

Use augmentation script.

Split data into `train` and `val` using **split-folders**. Make train and val folders first.

```
Input structure:
data
|_volumes
|_masks

Output:
data_splitted
|_train
  |_volumes
  |_masks
|_val
  |_volumes
  |_masks
```

Also need to `resize` to `(128, 128, 512).`

Use resize script.

Will resize beforehand for faster training. Also split into groups of 65.

Now creating groups of `64` slices with `create_64_slices_nifti.ipynb`.

Delete volumes with no foreground class.

Keras Method (again)

Not getting an yreasonable results.

Checked results. `loss` and `metric` values are weird and shows high `accuracy` but results are bad.

TorchIO approach (again)

Same as above.

Monai (again on new data)

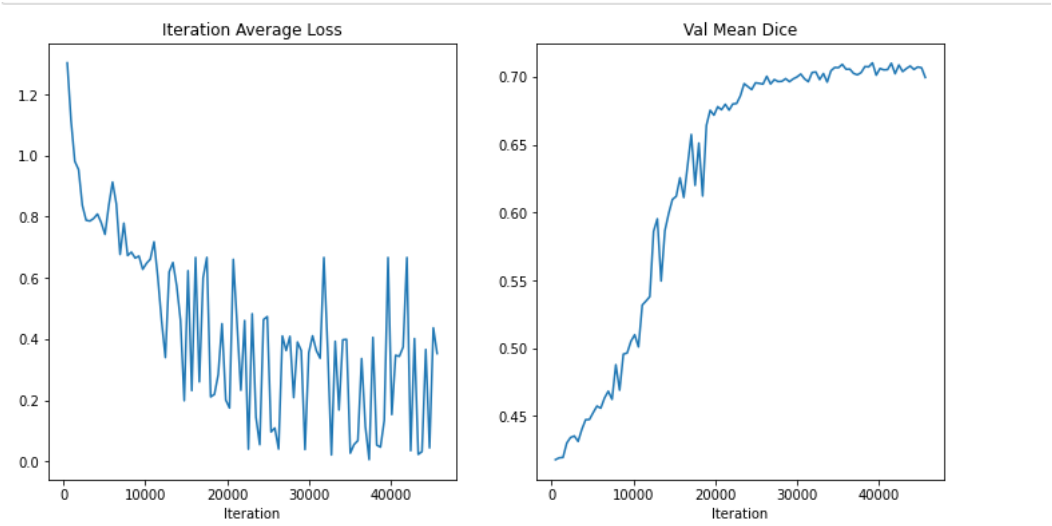
Installations

```
conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
pip install monai
pip install matplotlib
pip install torchio
```

Reference→ https://github.com/Project-MONAI/tutorials/blob/main/3d_segmentation/unetr_btcv_segmentation_3d.ipynb

Ran for `100 epochs`.

Got some better results. Dice goes up to 70%.



Need to check labels and classes now.

Found another issue. `Resize` transform changed input pixel values which caused more than 3 labels. Fixed it and training again now.

Also `Spacingd` and `CropForegroundd` were changing size of input images. Removed them as well.

Results were good. Dice went up to `85%` near 200 epochs which is really good. Also results on validation data were really good.

However on novel test data, it did not detect much.

Possible causes

Model not performing well

Missing pre-processing steps from torch io.

- Checking now
- Yup this was the reason.

Need to fix the `spacing` and `resize` before apply model

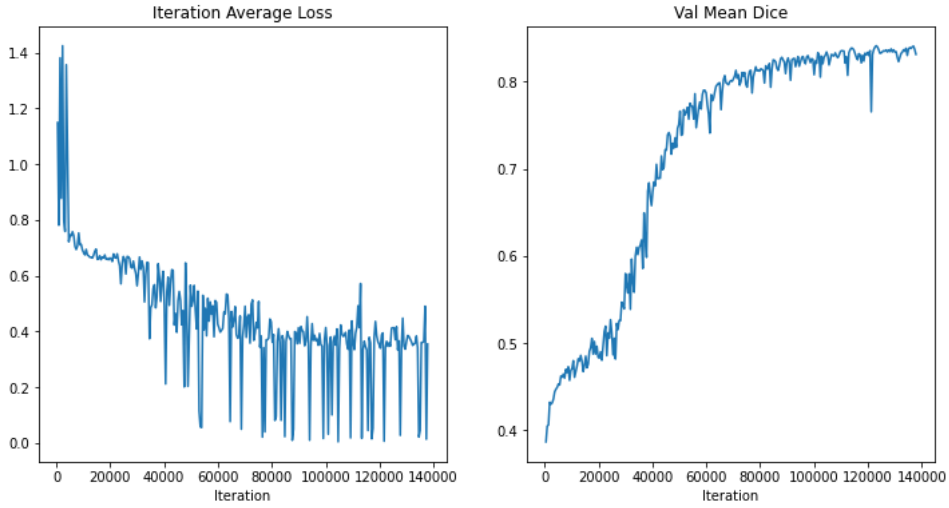
Steps for testing

- Preprocess volumes `spacing` and `resize` using **TorchIO**.
- Then make groups of 64 slices
- Then load with monai loader, apply transforms and predict.

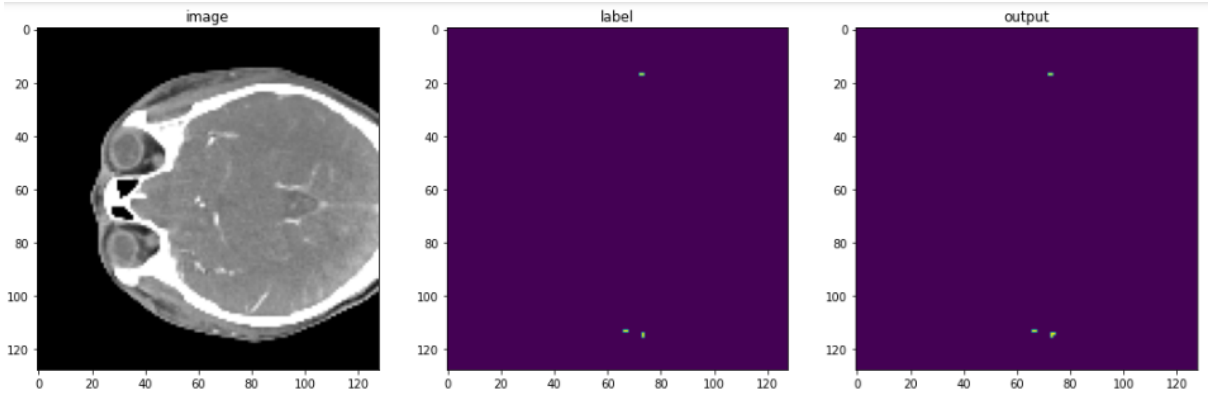
Results

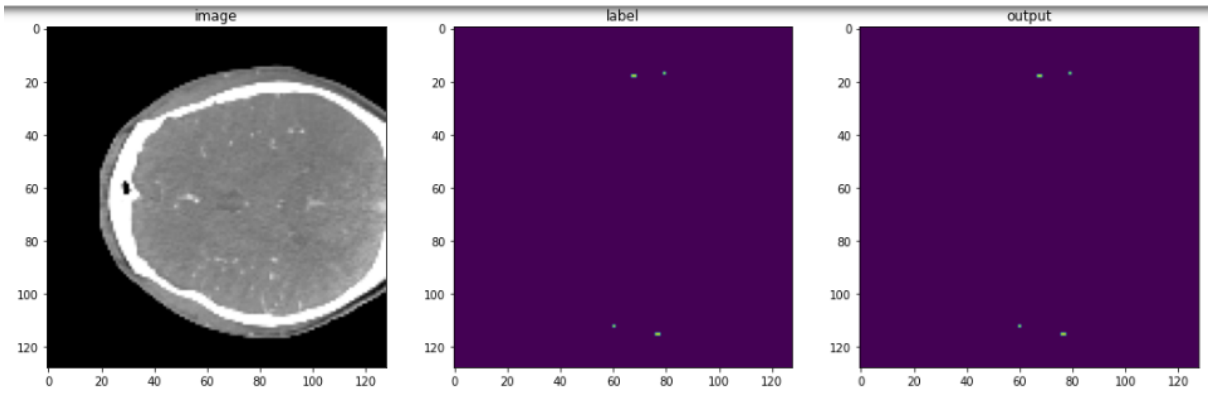
Model trained for 300 epochs

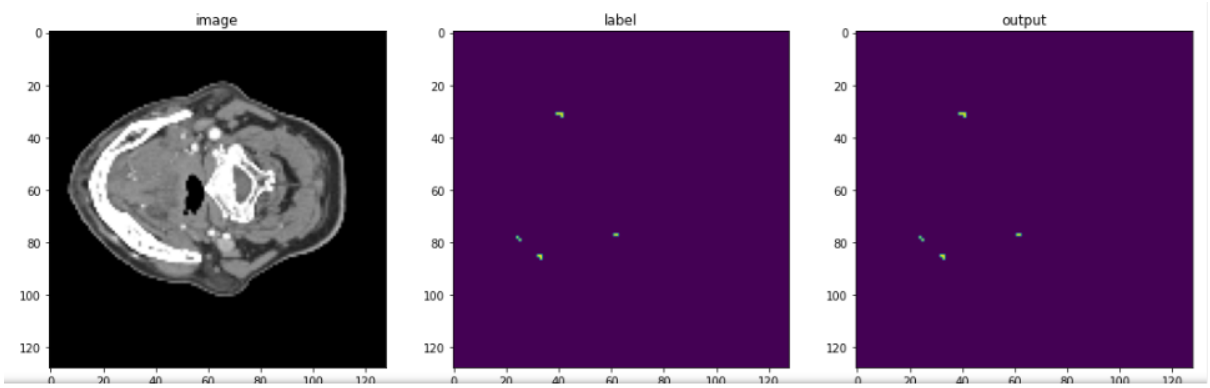
Looking good.

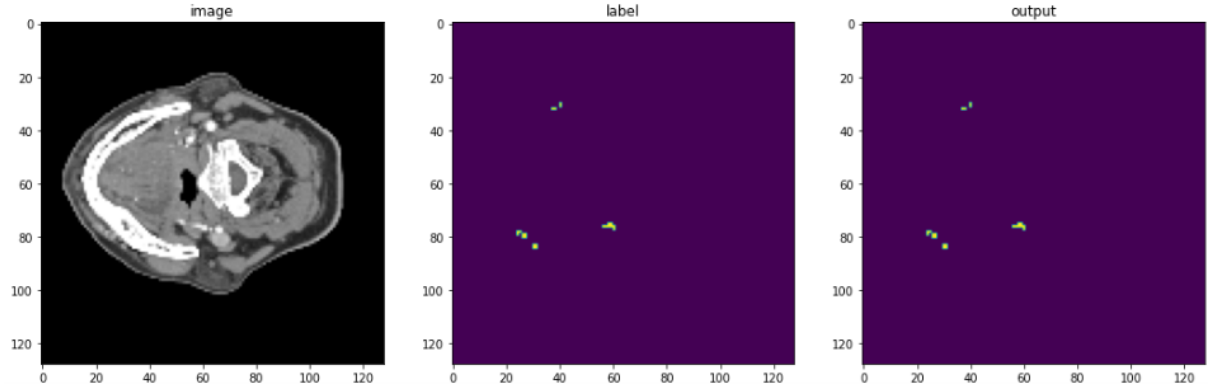


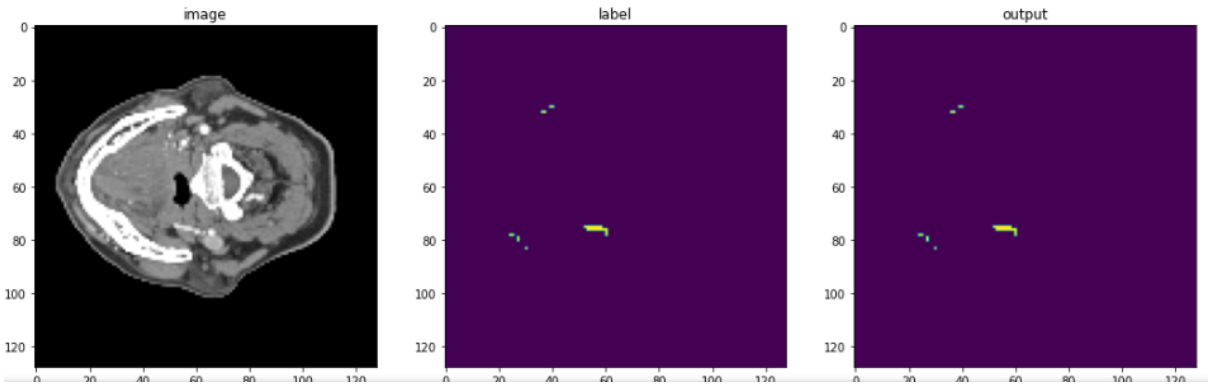
Validation Data Results











Test Data Results

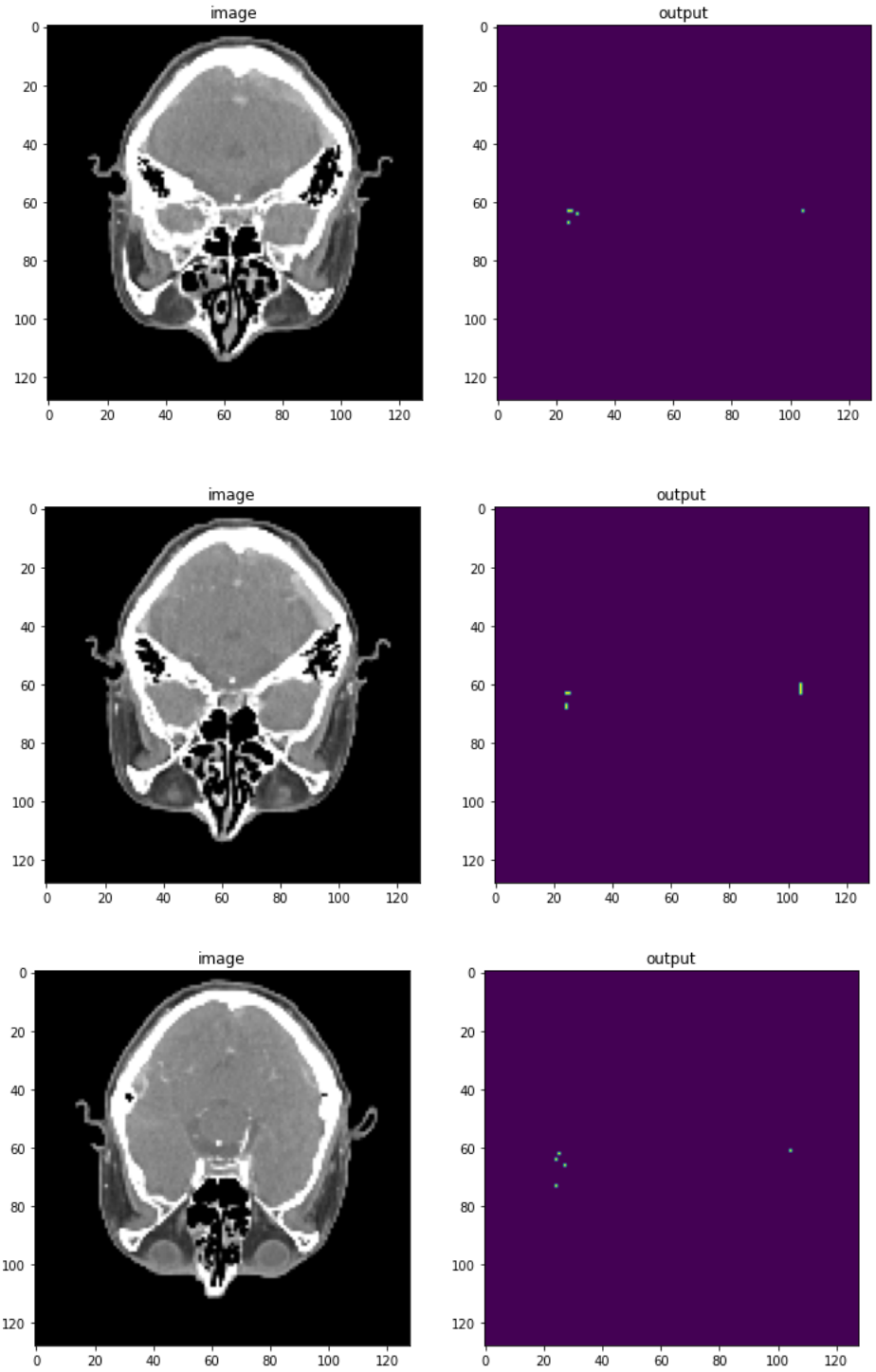
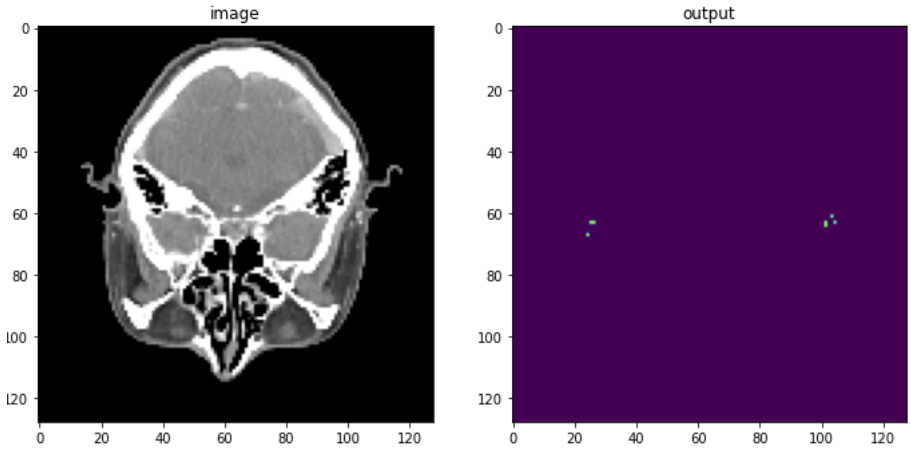


image
output
image
output
image
output



I have multiple models with and without `ce_wrights`.

There are very view `False Positives`.

Resizing to 512x512 from 128x128

```
pip install scikit-image
```

```
# Predict with fixed angle
import scipy.ndimage as ndimage
from skimage.transform import resize

angle = 90 # in degrees

case_num = 6  # 4, 6

with torch.no_grad():
    data = test_ds[case_num]
    img = test_ds[case_num]["image"]
    test_inputs = torch.unsqueeze(img, 1).cuda()
    test_outputs = sliding_window_inference(
        test_inputs, (128, 128, 64), 4, model, overlap=0.8
    )


    for i in range(0, 32):
        plt.figure("check", (18, 6))
        plt.subplot(1, 3, 1)
        plt.title("image")
        input_data = test_inputs.cpu().numpy()[0, 0, :, :, i]
        plt.imshow(ndimage.rotate(input_data, angle, reshape=True), cmap="gray")
        plt.subplot(1, 3, 2)
        plt.title("output")
        output_data = torch.argmax(test_outputs, dim=1).detach().cpu()[0, :, :, i]
        output_image = np.array(output_data)
        output_resized = resize(output_image, (512, 512, 1))
        plt.imshow(
            ndimage.rotate(output_resized, angle, reshape=True)
        )
        plt.show()
```

Above changed values for label map pixels.

Use TorchIO for resizing.

Didn't work well.

Using skimage again, but on 3D data.

```
# Predict with fresize

import scipy.ndimage as ndimage
from skimage.transform import resize

angle = 90 # in degrees

case_num = 6  # 4, 6

with torch.no_grad():
    data = test_ds[case_num]
    img = test_ds[case_num]["image"]
    test_inputs = torch.unsqueeze(img, 1).cuda()
    test_outputs = sliding_window_inference(
        test_inputs, (128, 128, 64), 4, model, overlap=0.8
    )

    output_data = torch.argmax(test_outputs, dim=1).detach().cpu()
    output_resized = resize(output_data, (1,512,512,64), anti_aliasing=False, order=0)

    for i in range(0, 32):
        plt.figure("check", (18, 6))
        plt.subplot(1, 3, 1)
        plt.title("image")
        input_data = test_inputs.cpu().numpy()[0, 0, :, :, i]
        plt.imshow(ndimage.rotate(input_data, angle, reshape=True), cmap="gray")
        plt.subplot(1, 3, 2)
        plt.title("output")
        output_image = output_resized[0, :, :, i]
        plt.imshow(
            ndimage.rotate(output_image, angle, reshape=True)
        )
        plt.show()
        print(np.unique(output_image, return_counts=True))
```

`order=0` important to not introduce new labels.

Saving as nifti

```
data = output_resized
data_array = np.array(data, dtype=np.float32)
print(data_array.shape)

data_reshaped = data_array.transpose(1,2,3,0)
print(data_reshaped.shape)

import nibabel as nib
img = nib.Nifti1Image(data_reshaped, np.eye(4))

nib.save(img, './monai_data_directory/results/output.nii.gz')
```

Save as nrrd

```
pip install pynrrd
```

```
data = output_resized
data_array = np.array(data, dtype=np.float32)
print(data_array.shape)

data_reshaped = data_array.transpose(1,2,3,0)
print(data_reshaped.shape)

data_flip = np.flip(data_reshaped, axis=0)
data_flip = np.flip(data_flip, axis=1)

import nrrd
nrrd.write('./monai_data_directory/results/output.nrrd', data_flip)
```

Whole volume prediction

Combine predictions from sub-volumes to one volume.

```
image = []
for patient in test_patients:
    test_image_list = sorted(glob.glob(patient + '/*.nii.gz'))
    test_files = [{'image': image_name} for image_name in test_image_list]
    test_ds = CacheDataset(data=test_files, transform=test_transforms, cache_num=10, cache_rate=1.0)
    test_loader = DataLoader(test_ds, batch_size=1, shuffle=False, pin_memory=True)

    with torch.no_grad():
        for case_num in range(len(test_image_list)):
            data = test_ds[case_num]
            img = test_ds[case_num]["image"]
            test_inputs = torch.unsqueeze(img, 1).cuda()
            test_outputs = sliding_window_inference(
                test_inputs, (128, 128, 64), 4, model, overlap=0.8
            )

            output_data = torch.argmax(test_outputs, dim=1).detach().cpu()
            output_resized = resize(output_data, (1,512,512,64), anti_aliasing=False, order=0)

#             for i in range(0, 64):
#                 plt.figure("check", (18, 6))
#                 plt.subplot(1, 3, 1)
#                 plt.title("image")
#                 input_data = test_inputs.cpu().numpy()[0, 0, :, :, i]
#                 plt.imshow(ndimage.rotate(input_data, angle, reshape=True), cmap="gray")
#                 plt.subplot(1, 3, 2)
#                 plt.title("output")
#                 output_image = output_resized[0, :, :, i]
#                 plt.imshow(
#                     ndimage.rotate(output_image, angle, reshape=True)
#                 )
#                 plt.show()

            data = output_resized
            data_array = np.array(data, dtype=np.float32)
            #data_reshaped = data_array.transpose(1,2,3,0)
            data_flip = np.flip(data_array, axis=1)
            data_flip = np.flip(data_flip, axis=2)
            image.append(data_flip)
    break
```

Tried predicting on volumes and appending results into one array which result in an array like `[8,1,512,512,64]` → shows 8 subvolumes each of size `512x412x64`.

Then I reshaped it using numpy.reshape() but results were not right.

```
image = np.array(image)
print(image.shape)

# Save groups as nrrd
import nrrd
for i in range(image.shape[0]):
    print(np.unique(image[i,0,:,:,:], return_counts=True))
    nrrd.write(f'./monai_data_directory/results/output_groups/{i}.nrrd', image[i,0,:,:,:])
```

```
temp = image.transpose(2,3,4,1,0)
output = temp.reshape(512,512,512)
print(output.shape)
nrrd.write(f'./monai_data_directory/results/output_whole/output_whole.nrrd', output)
```

This method worked:

```
arr = np.empty((512,512,0))
for i in range(0, temp.shape[0], 1):
    arr1 = np.array(temp[i,0,:,:,:])
    arr = np.concatenate((arr,arr1), axis=2)
```

```
print(arr.shape)
print(np.unique(arr, return_counts=True))

nrrd.write(f'./monai_data_directory/results/output_whole/test.nrrd', arr)
```

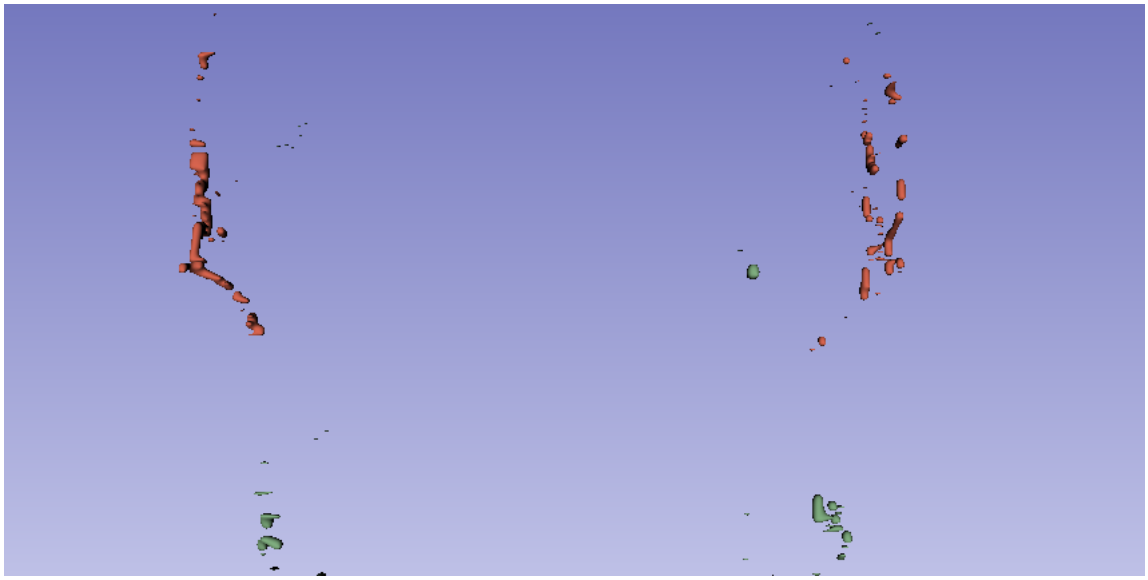
Final way:

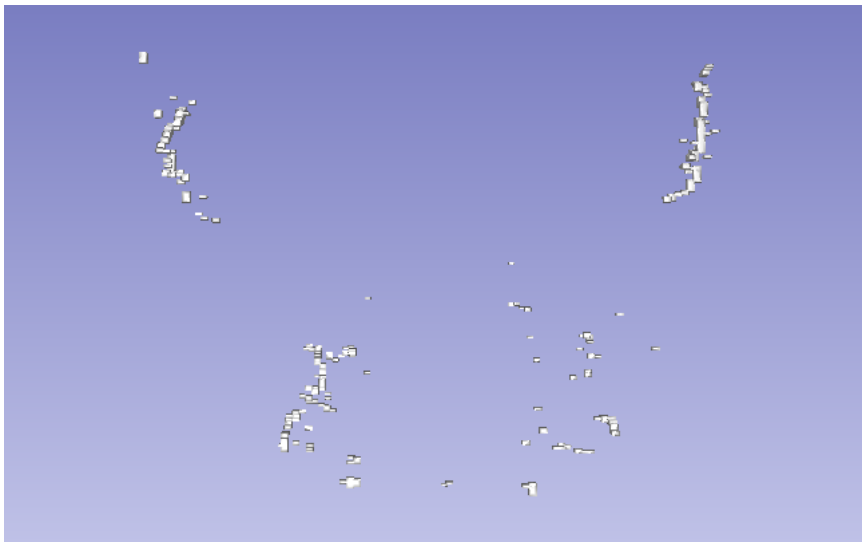
```
image = np.array(image)
print(image.shape)

# Save groups as nrrd
import nrrd
for i in range(image.shape[0]):
    print(np.unique(image[i,0,:,:,:], return_counts=True))
    nrrd.write(f'./monai_data_directory/results/output_groups/{i}.nrrd', image[i,0,:,:,:])

# Save as one volume
arr = np.empty((512,512,0))
for i in range(0, temp.shape[0], 1):
    arr1 = np.array(temp[i,0,:,:,:])
    arr = np.concatenate((arr,arr1), axis=2)

print(arr.shape)
print(np.unique(arr, return_counts=True))

arr = np.flip(arr, axis=1)

nrrd.write(f'./monai_data_directory/results/output_whole/output_whole.nrrd', arr)
```

Make it automatic later.

- Convert data into groups
- Predict
- Convert back to one volume

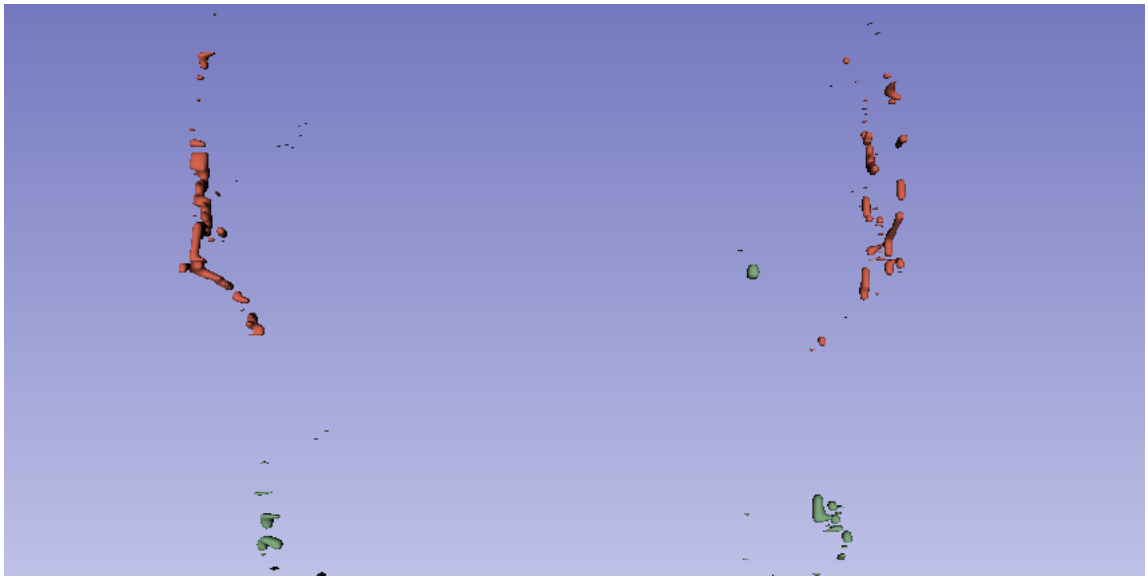
We can then use `Thresholding` or `Volume Rendering` in 3D Slicer to get 3D view.

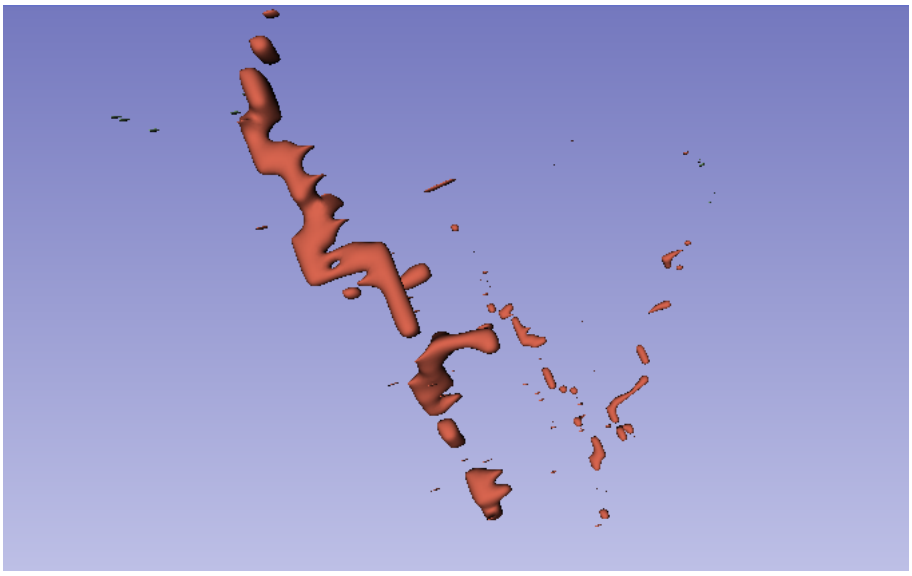




Issue

The output seems disconnected.





There can be various reasons like

- Using 128x128 instead of 512x512
- Using 64 slices sub-volumes
- Difficult task
- Labeling issues

I can try to use slice overlap in subvolumes.

Also, I will try simple UNet now.

Simple UNet

Dice is around 75%. Not as good as UNETR.

UNETR 1000 epochs

Dice goes up to 85%.

Disconnected prediction issue

Original labels from Heejin are good.

My labels after **TorchIO** transformations are also good. Issue happened after that.

Issue happened during `128x128x512` resize woth TorchIO to make the size same. Results are good with `512x512x512`.

Need to check other arguments or monai transform method.

Changed interpolation mode, but not helpful.

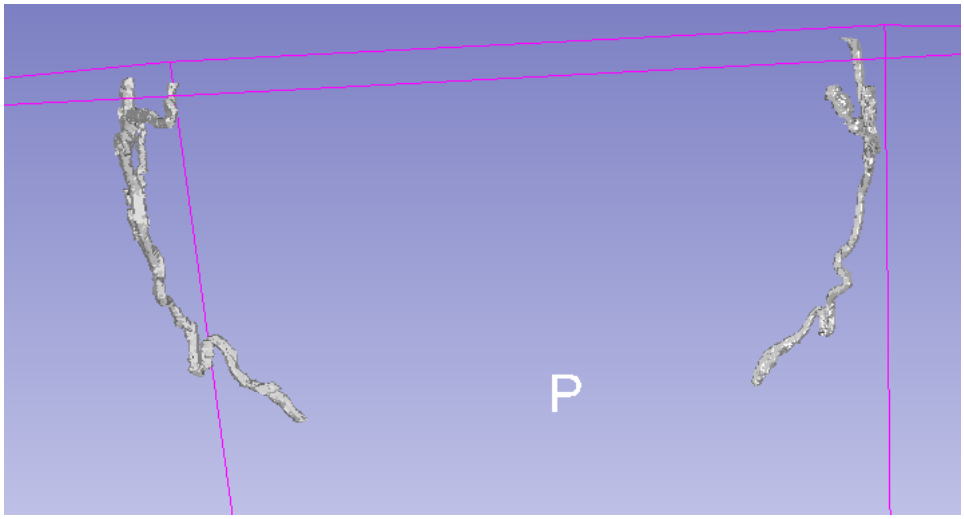
Trying to find a resize method.

Monai method is crashing, dunno why.

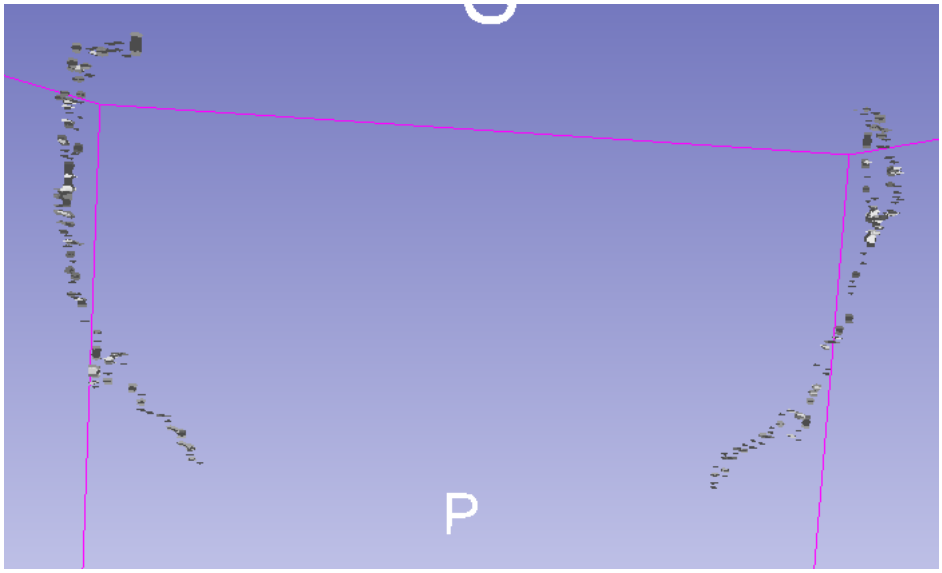
Skimage needs `order=0` and `anti_aliasing=False` to keep label values same. Results not good.

Maybe I need to use 512x512x16 instead of 128x128x64.

Before resize



After resize



How to resolve?

check resampling with different arguments.

Try 512x512x16 with overlap of 4.

512x512x16 with MONAI

Make dataset first.

```
Original data → 16 patients
Augmented → 89 Train, 23 val
Grouped → 3738 Train, 966 Val
After cleaning -> 1991 Train, 513 Val
```

Cannot use `CacheDataset` due to large data size.

Training is slower now.

Don't use `slice overlap` in test volumes.

Trained model for 2 days for 100 epochs. Dice was 80.5% at the end.

Preprocess train data with `spacing` and `resize`. Then make groups and predict.

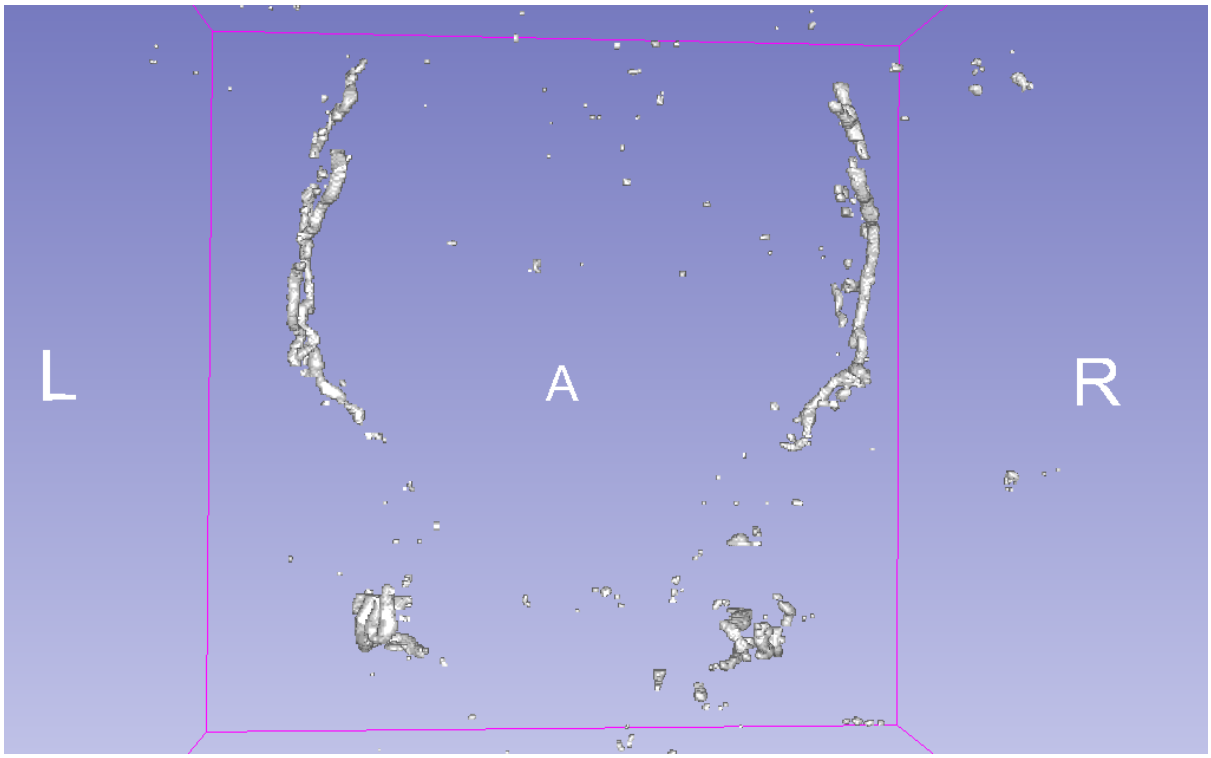
- Use `test_preprocessing` script
- Then `test_data_make_groups`
- Then use `unetr_multiclass_prediction_512x512x16`

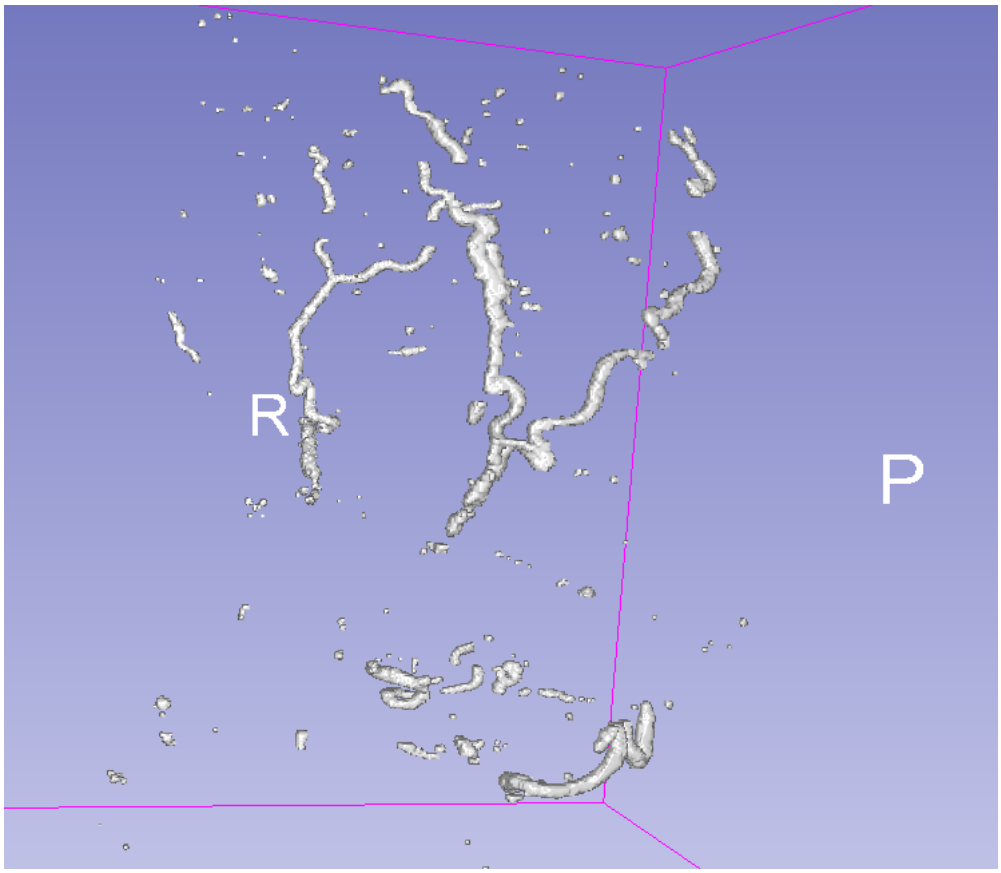
Loss Testing

DiceCELoss

Used weights [1,1000,1000]

Results are better now but there is some noise.

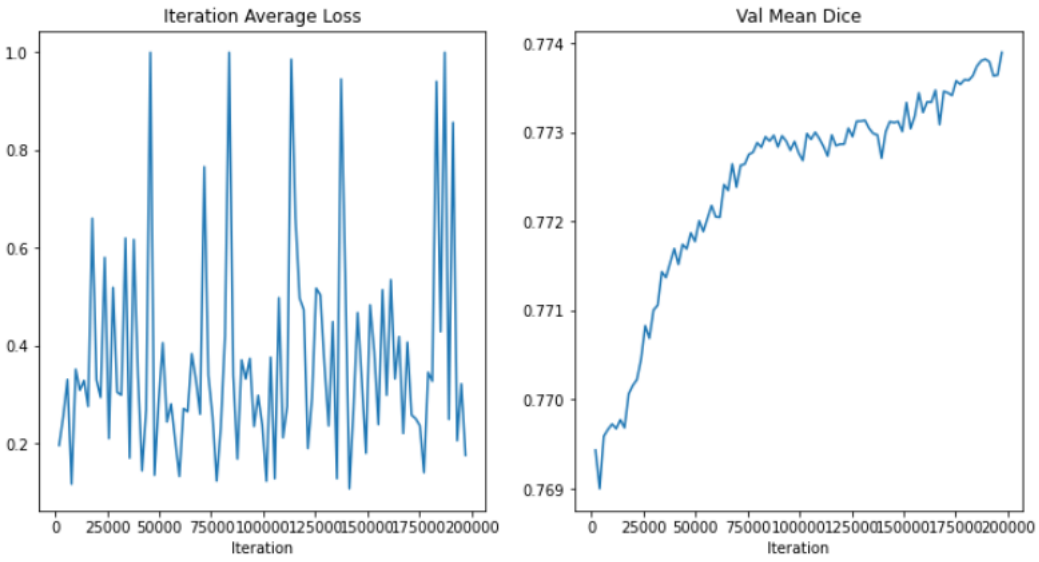


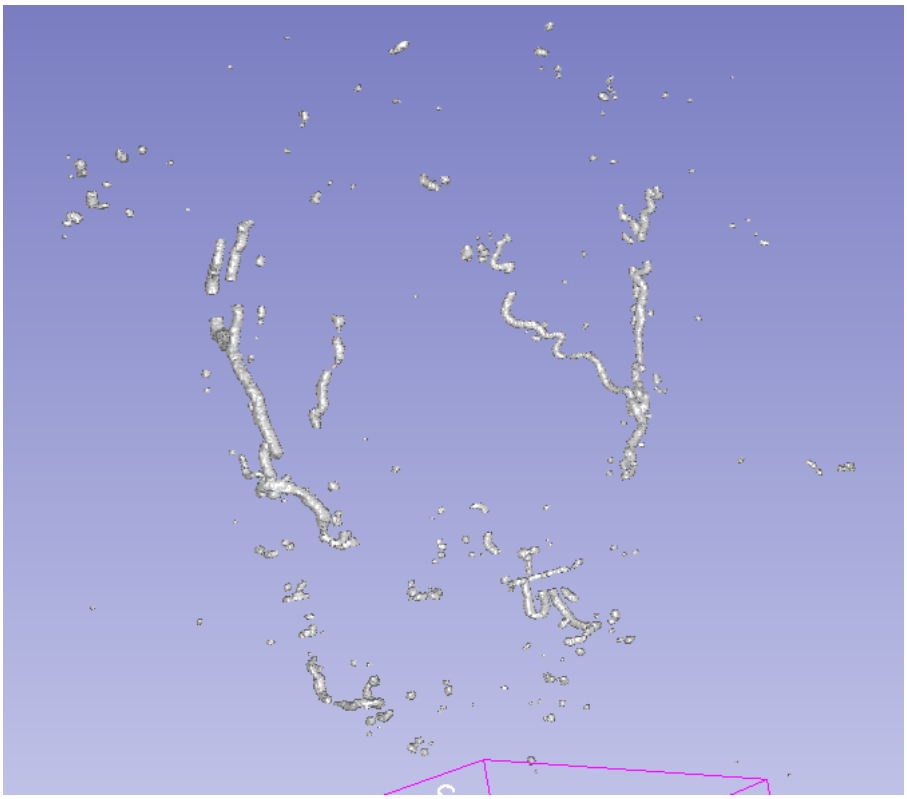


GeneralizedDiceLoss

Trained for 150 epochs.

Dice didn't increase much. `77.39` max and also it increases extremely slow.





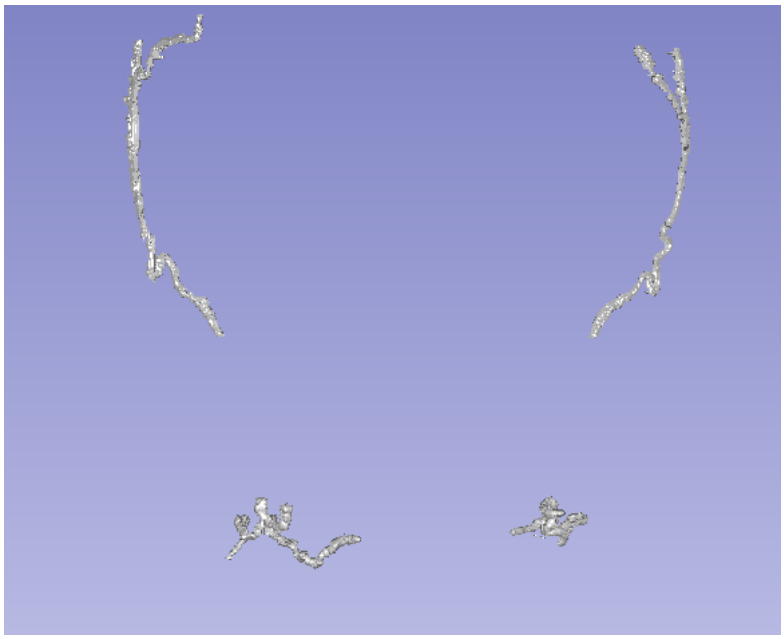
Still has noise.

Skimage Resizing

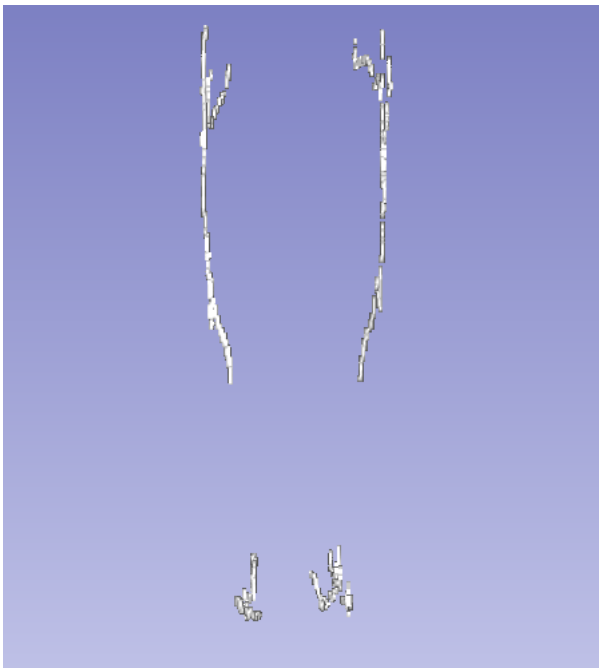
Image size is issue → results in large training time.

`skimage.transform.resize` is working.

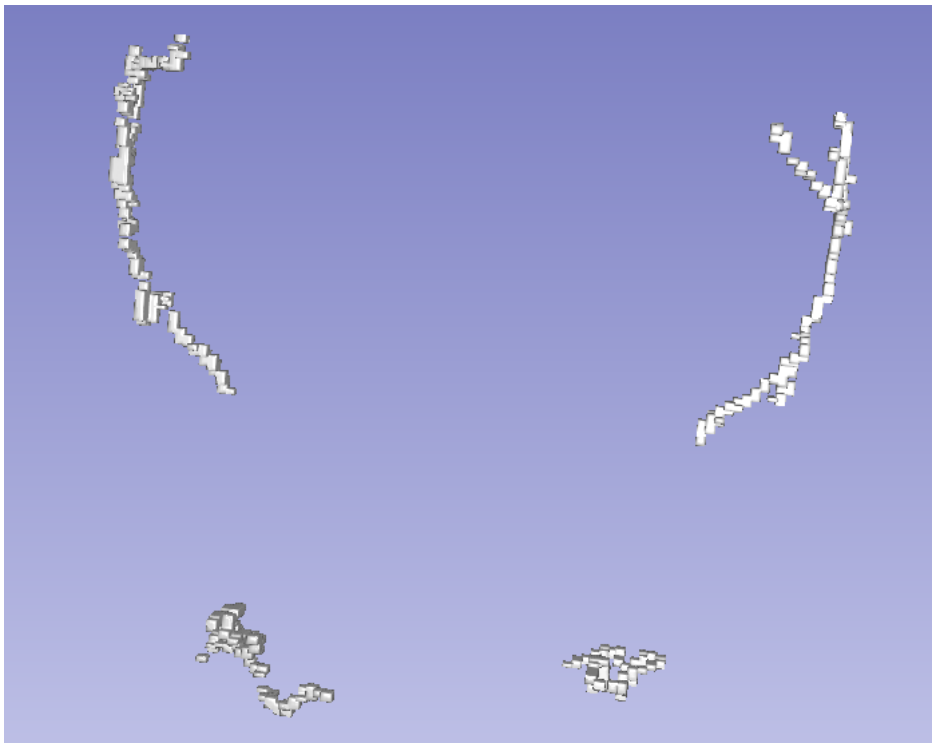
Before Resize



After resize to 128x128x512



After resizing back to 512x512x512



Still it is not perfect.

```
import os, glob
import nibabel as nib
import skimage.transform as sktrans
import numpy as np

label_path = '7_spacing.nii.gz'

def read_nifti_file(filepath):
    """Read and load volume"""
    # Read file
    scan = nib.load(filepath)
    # Get raw data
    scan = scan.get_fdata()
    return scan

label = read_nifti_file(label_path)
print(label.shape)
print(np.unique(label, return_counts=True))

# Resize to 128x128x512
result = sktrans.resize(label, (128,128,512), order=0, preserve_range=True, anti_aliasing=False)
print(result.shape)
print(np.unique(result, return_counts=True))

result = nib.Nifti1Image(result, np.eye(4))
nib.save(result, '7_skimage_resize.nii.gz')

# Interpolate to 512x512x512
label_resized = read_nifti_file('7_skimage_resize.nii.gz')
print(label_resized.shape)

interpolated = sktrans.resize(label, (512,512,512), order=0, preserve_range=True, anti_aliasing=False)
print(interpolated.shape)
print(np.unique(interpolated, return_counts=True))

interpolated = nib.Nifti1Image(interpolated, np.eye(4))
nib.save(interpolated, '7_skimage_interpolated.nii.gz')
```

Disconnection occurs during the downsampling step which is resonates in the upsampling step as well.

Effect of anti_aliasing

`anti_aliasing` applies a **Gaussian filter**. That makes it smooth somehow and disconnection does not happen.

However, it imposes another issue. The surrounding pixels are also considered as white areas due to blur. Also the label value range changes from 0,1,2 to many interpolated values.

Solution can be to define some `thresholds` and remap values back to 0,1 and 2.

First I will try to find another resize method. If nothing works then I should try `thresholding` with `skimage.resize`.

Creating and training with 256x256x32 dataset

Another idea, less or no disconnections in this case.

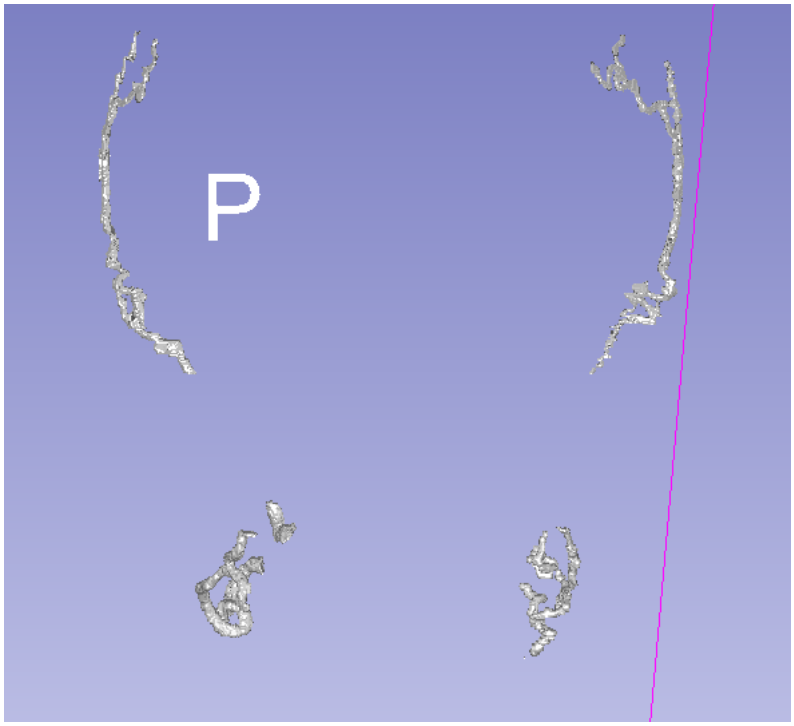
Do `resample` so that if some data has different space, it becomes consistent.

Will convert data to `256x256x32` and then to groups of 32 slices.

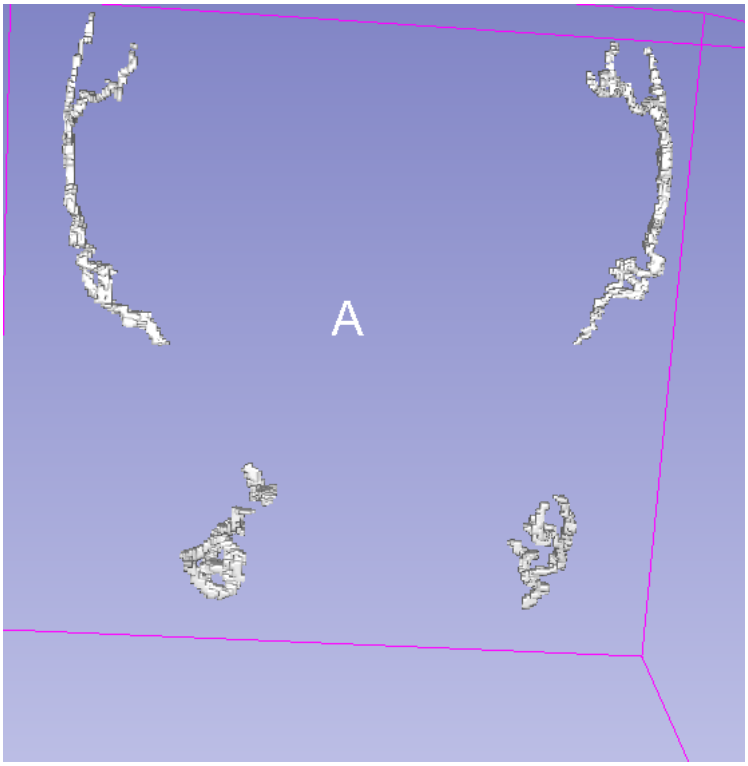
- Resample
- Resize
- Divide to groups
- Remove empty groups

Using `skimage.transforms.resize`.

Before resize [575,575,512]



After downsampling [256x256x512] and upsampling[512x512x512]




```
size of dataset

Train (before removing empty) -> 1602
Val (before removing empty) -> 414
```

```
Train(after) -> 902
Val(after) -> 234/235
```

I trained and for some reason there is issue. `Dice` is 96% in first epoch which shows there is an issue.

I used `skimage` for resizing to `256x256x512` and maybe that caused the issue. Now using `torchio`.

So `skimage` resize was the issue. Now `dice` is ok.

> 💡 In the MONAI examples they used `RandCropByPosNegLabeld` for training dataset only and not for validation. I should test and see what it does.

> 💡 Use `RandCropByPosNeg` with 512x512x128 sub-volumes? i.e. initial resize to `512x512x512` and then make `four` sub-volumes for each patient. Then use `RandCrop` function during training with size of `128x128x128`

I want to try this above idea first. So I will do that now.

Creating and testing with RandCropByPosNeg

Create dataset of 512x512x128.

Check without overlap first and then maybe with overlap.

- Resampling to `(0.4,0.4,0.5)` and resizing to `512x512x512`
- Creating sub-volumes of `512x512x128` and removing empty

```
Train -> 267
Val -> 69
```

Ran into CUDA memory issue with 128x128x128. Maybe I do not need to resize to `512x512x512`. I can just use `RandCrop` with `(96,96,96)` directly.

Testing without any `spacing resample`, `resize` and `sub-grouping`.

Following example

Link → https://github.com/Project-MONAI/tutorials/blob/main/3d_segmentation/unetr_btcv_segmentation_3d.ipynb

Following example with `RandCropByPosNegLabeld`, `CropForegroundd` and `Spacingd` along with other transforms.

Original image size is maintained and during training, patches of `96x96x96` are used with `RandCropByPosNegLabeld` function.

`Spacingd` helps reducing image size.

💡 `pixdim` value in spacing? Try different values?

It is not working. The `cropping resize` methods.

Back to 128x128x64

Going back to `128x128x64` but this time online resizing with MONAI instead of offline with torchio.

- Take data with `spacing` fixed to (0.4, 0.4, 0.5) and `resized` to 512x512x512.
- Grouping (64 slices) and cleaning.

```
Train -> 467
Val -> 122
```

It seems like model is not training again. So I printed label values.

```
In [10]: print(np.unique(val_ds[case_num]["label"], return_counts=True))
```

```
(array([0.    , 0.0625, 0.125 , 0.1875, 0.25  , 0.3125, 0.375 , 0.4375,
        0.5   , 0.5625, 0.625 , 0.6875, 0.75  , 0.8125, 0.875 , 0.9375,
        1.    , 1.125 , 1.25  , 1.375 , 1.5   , 1.625 , 1.75  , 1.875 ],
       dtype=float32), array([1047656,      72,     126,      70,      77,      39,      61,
             32,      60,      39,      43,      28,      46,      16,
             35,      31,      90,       9,       8,       7,       5,
              4,       9,      13]))
```

Some `transformation` is changing values. Maybe same thing happened in the above example case.

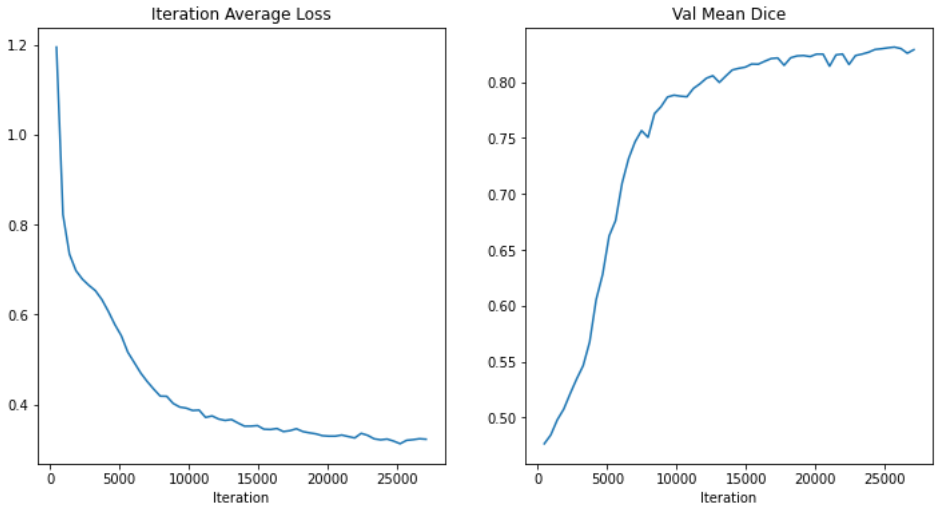
`Resize` transform was causing the issue. `mode` parameter was not set. I set it to `nearest-exact` and now label values are fine.

Training with `Adam`. Check `AdamW` later.

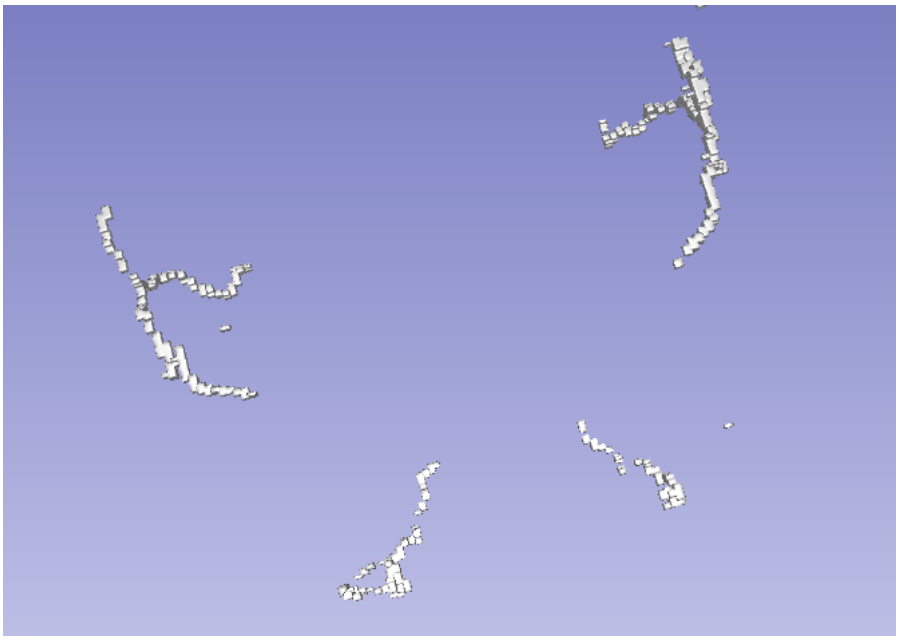
Trained UNet mode. Dice was high but test data predictions are bad.

Number of labels in `validation` data prediction is higher and looks like a vessel. But on `test` data it predicts less points and results are bad.

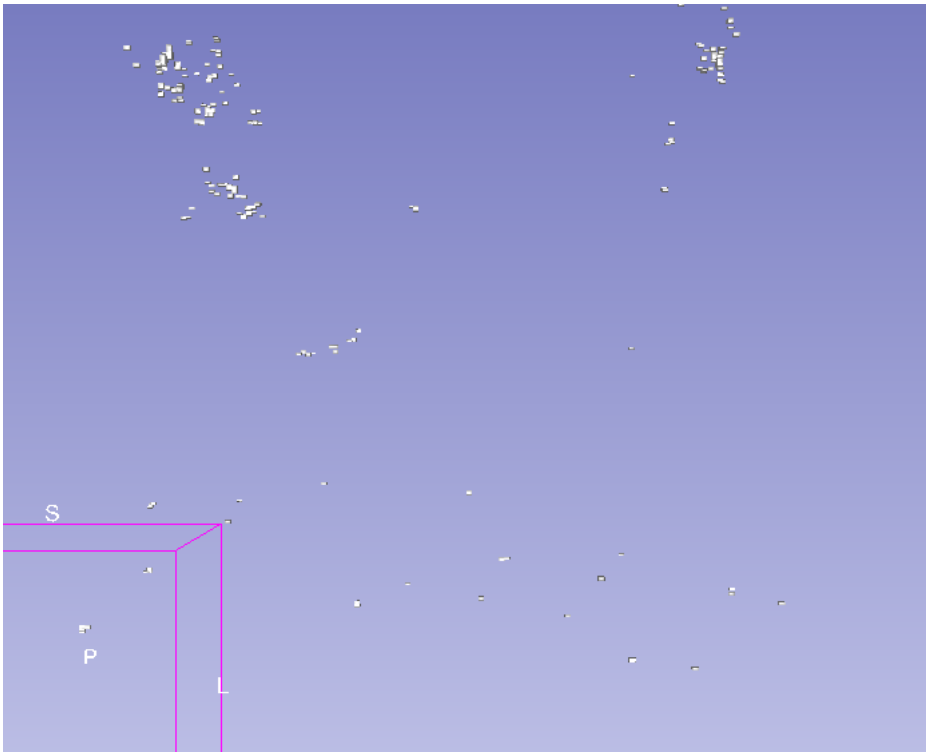
- UNET can be bad
- 128x128x64 can be bad



Validation



Test



Model is not generalizing well to new data. Validation data results are good because validation data is same as test data. Split is example based and not patient-based.

Thoughts on data split

Data split should be on subject-level and not volume-level. But I do not have enough data to do so. If I do subject-level then my validation set is too tiny.

On the other hand, volume-level split does not give much idea about model generalization.

Let's split data `patien-wise`.

First try without augmentation and then with augmentation. Use `UNETR` model with different input sizes like 128x128x64, 512x512x16, 256x256x64 etc.

Main Issues Faced

- Big volume size
  - Working with patches
- Loss function issue
  - Dice loss needs same `target` and `output` size
    - But some `target` don't have one label so shape differs
  - Trying BCE now
  - Maybe issue is because of `one-hot` encoding of the `target` labels
- Different physical space for input volumes
  - Possible solutions `tio.ToCanonical()` and `tio.Resample()`

- `origin` missaligment issue. Fixed with `registration` in **3D Slicer**.

*****Research Phase II*****

- Starting with new data split. Will divide the data patient-wise to get true results without biasing.
- Will add additional data
- Try without augmentation and then go for augmentation if required.

Steps

- Make dataset folder i.e. `train` and `test`.
- Fix the labels using `label_swap.ipynb` and `label_mismatch_fix.ipynb`
  - Separated files which have reverse labels. Fix them!

```
# Fix the labels
for mask in glob.glob(path + '/*.nii.gz'):
    name = mask.split('/')[-1]
    label = read_nifti_file(mask)
    label[label==2] = 999  # placeholder value
    label[label==1] = 2
    label[label==999] = 1
    label_arr = nib.Nifti1Image(label, None)
    nib.save(label_arr, os.path.join(out_path, name))
```

  - 6, and 18 had label mismatch issue (one extra label). Fixed that.

```
for mask in issue_masks:
    path = os.path.join(labels_dir, mask)
    label = tio.LabelMap(path)
    print("Before fix: ", np.unique(label.data))
    label.data[label.data==3] = 2
    print("After fix: ", np.unique(label.data))
    label.save(os.path.join(out_dir, '{}.nii.gz'.format(mask)))
```

  - The above step caused issue in one file `18.nii.gz`. Maybe I should fix extra label first and then swap. Or I should just also check manually after fixing to verify. Leter I fixed `18.nii.gz` manually.

- Split into `val` and `train` using `data_split.ipynb`
  - train → 37, val → 7 (85, 15 split)

- Skipping `preprocessing` step where I fixed `spacing` and `resized` to `512x512x512`. No need to do that. Sub-grouping will take care of the size.
  - If this causes an issue, then I will include it.
- Create sub-volume groups
  - Creating 16 slices for now (input will be 512x512x16)
  - Overlap of 4
- Remove empty sub-volumes with `remove_empty_volumes.ipynb`

Include train time augmentations like MONAI. Compare with and without.

Not TorchIO augmenting data to increase size for now. It might not be that useful and increases training time.

Test with and without

- CropForeground
- MONAI augmentation
- TorchIO augmentation

Checking reverse labels again.

```
for mask in glob.glob(path + '/*.nii.gz'):
    name = mask.split('/')[-1]
    label = nib.load(mask).get_fdata()
    unique = np.unique(label, return_counts=True)
    if unique[1].shape[0] > 2:
        if unique[1][1] < unique[1][2]:
            print(name)
            print('\n')
```

The above check is not completely reliable. In some cases maybe label 2 is more abundant than label 1. So I am checking it manually now.

I feel like there is still some label issue with dataset. Checking the masks manually.

```
Issue files

18.nii.gz
```

Maybe I messed it up while fixing labels. Copying it again from database and fixing. Then will make groups of it and add to `train` data after removing the empty ones.

UNETR seems to be stuck when loss is used without weights.

Trying different losses and sizes but not working.

Training with `GeneralizedDiceFocalLoss` and input size of `256x256x16`. Validation size is not changed and `sliding_inference` is used.

Dice is increasing extremely slowly.

Tried `RandCrop` with small patches but no luck.

Trained a model over the weekend with `256x256x16` resize but results are not good.

Last time data was split after augmentation. This time before.

Trained UNet with `TverskyLoss` but Dice went up to 57% after 100 epochs. Input resized to 256x256x16.

Creating augmented dataset.

- Split patients into train and val

- Do augmentation (5 per patient)
- Divide into groups and clean

Trained with `GeneralizedDiceLoss` and `DiceCELoss` on previous data (without new data). It is working.

For old dataset

```
len(train_files) // 6
len(val_files) // 2

for CacheDataset.
```

Figured out the issue

```
weights_path = os.path.join(root_dir, "best_metric_model.pth")
if os.path.exists(weights_path):
    checkpoint = torch.load(weights_path)
    model.load_state_dict(checkpoint)
else:
    print("No checkpoint found.")
```

The above code was inside the `train function` which caused the issue.

It should be outside the function.

*****Research Phase - III*****

Back to new data now.

Using without `TorchIO` augmentation.

```
Train: 37 patients -> 548 sub-volumes
Val: 7 patients -> 101 sub-volumes
```

Approach 1

Using patches of (96,96,96) from original data with `DiceCELoss`

Did not work.

Approach 2

Using `512x512x16` sub-volumes with resize of `256x256x16`.

Remember we do not change size of the validation data. We use `sliding_window_inference` method.

No augmentation.

DiceCELoss

Not learning without `ce_weight`. Need to use it.

Takes time to start converging.

Dice goes up to 60%.

GeneralizedDiceLoss

Training seems unstable.

Convergence is slow and unstable.

Dice goes up to 60%.

Tversky loss

Not working well.

Dice not increasing fast.

Unstable maybe.

GeneralizedDiceFocalLoss

Not working well.

Dice not increasing fast.

Also tried UNet and Unetr both.

Size of `256x256` gives best dice up to 60%. Which is not high.

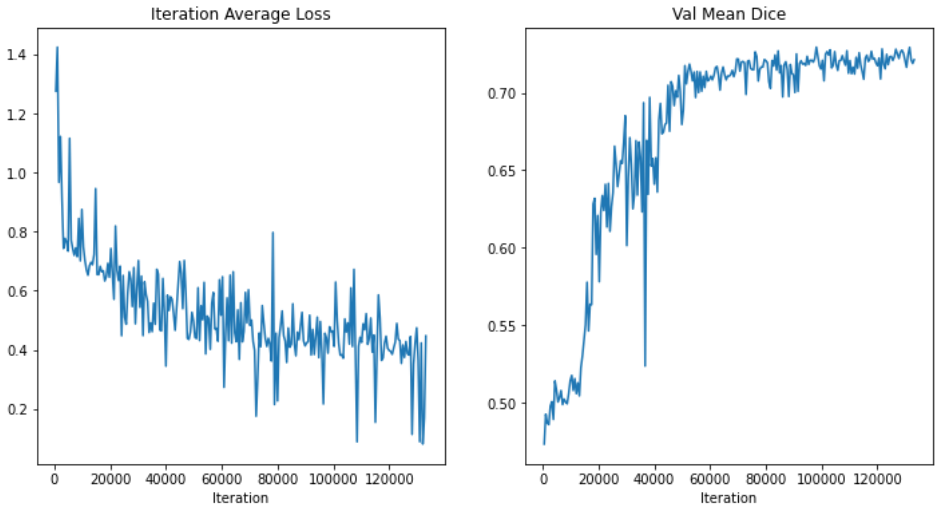
Approach 3

Input size: 512x512x16

UNETR architecture.

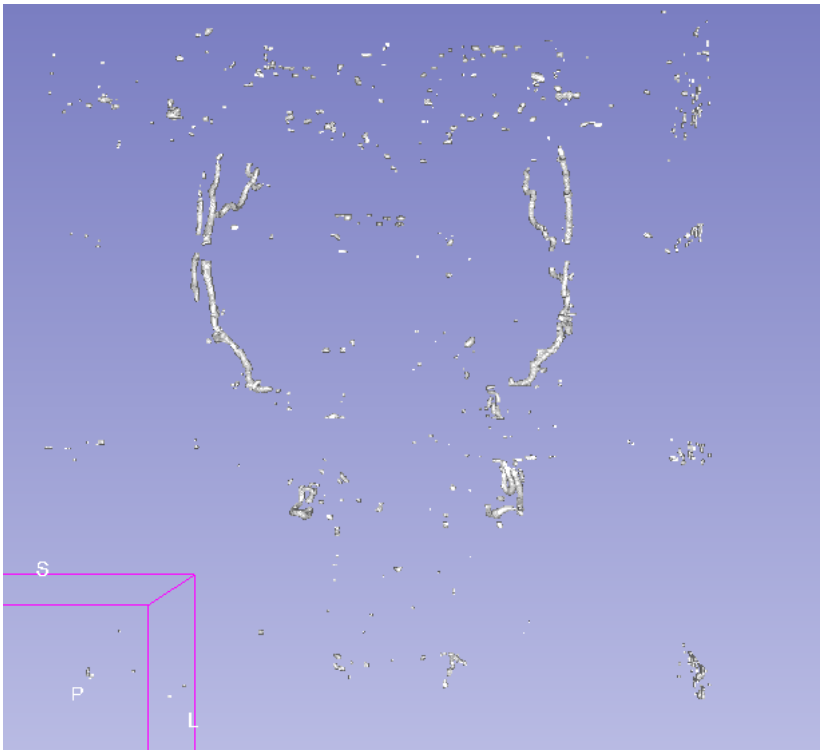
DiceCELoss

ce_weight [1,1000,1000]



Dice goes up to 73%.

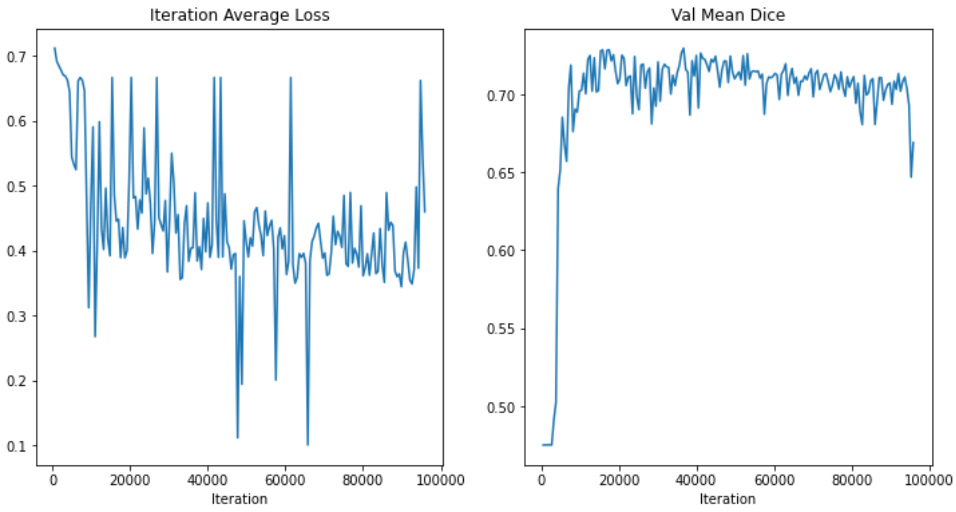
Predictions not good. Seems like model is confusing labels as well between two classes and also not predicting all the labels.



A lot of noise → false positives

Tversky Loss

alpha=0.7, beta=0.3



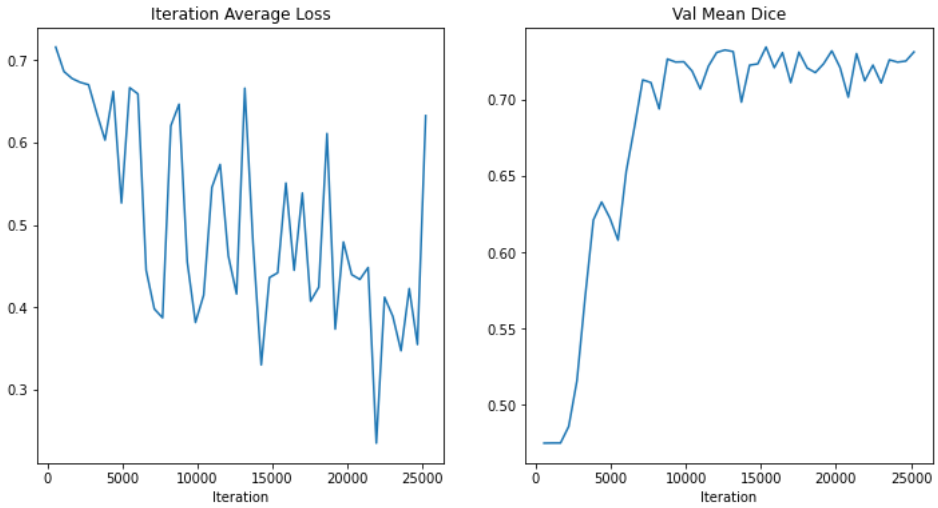
Converges faster.

Dice up to 73%.

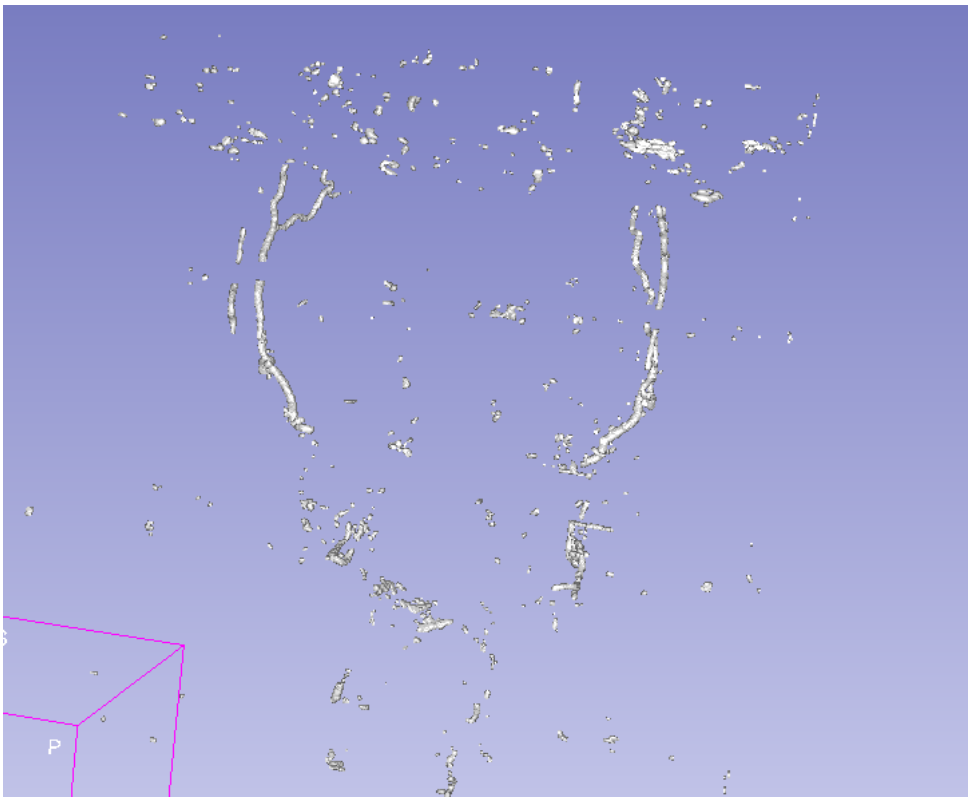
But it is giving less predictions i.e. `recall` is low. Need to tune `alpha` and `beta`.

Predictions not good. Seems like model is confusing labels as well between two classes and also not predicting all the labels.

alpha=0.5, beta=0.5

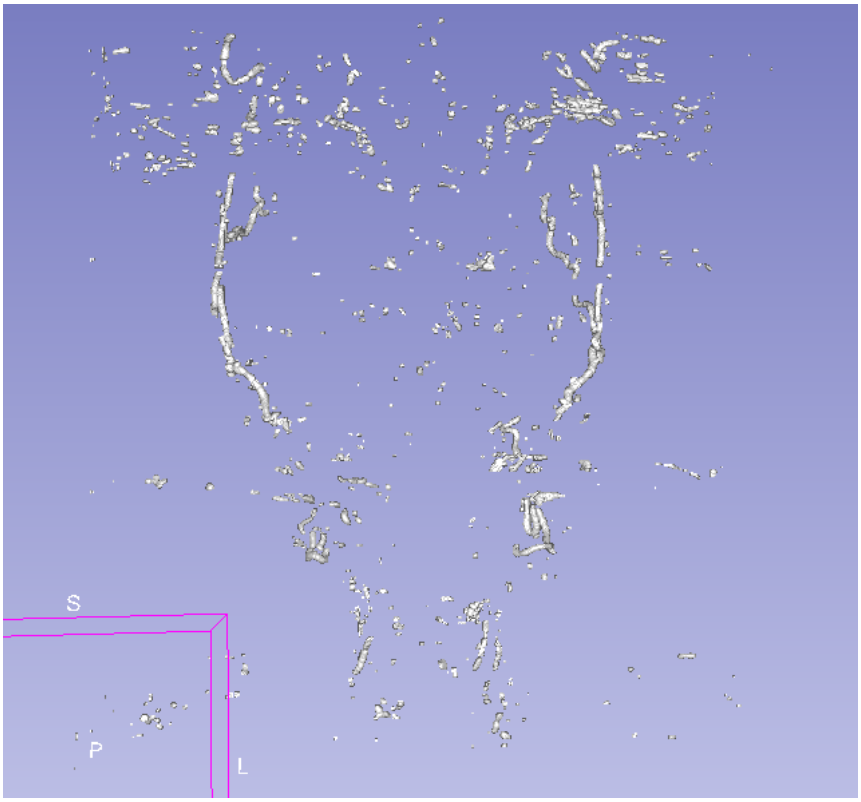


Dice: 73.43%

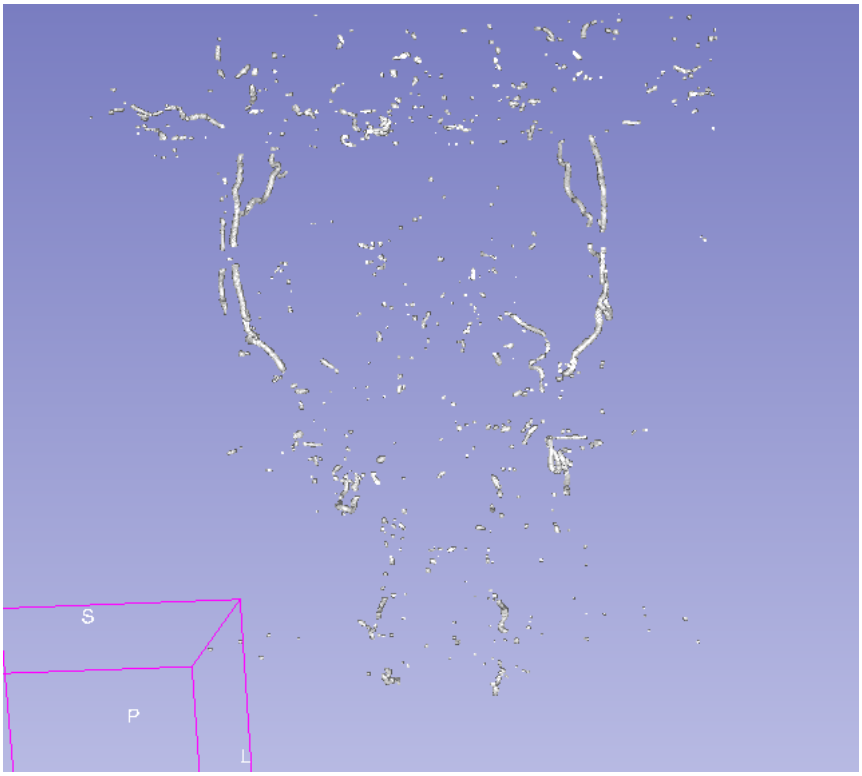


A lot of noise.

alpha=0.3, beta=0.7



alpha=0.8, beta=0.2



Approach 4

Training with augmented data.

Taking too long and `Dice` is increasing slowly.

Ensemble Learning

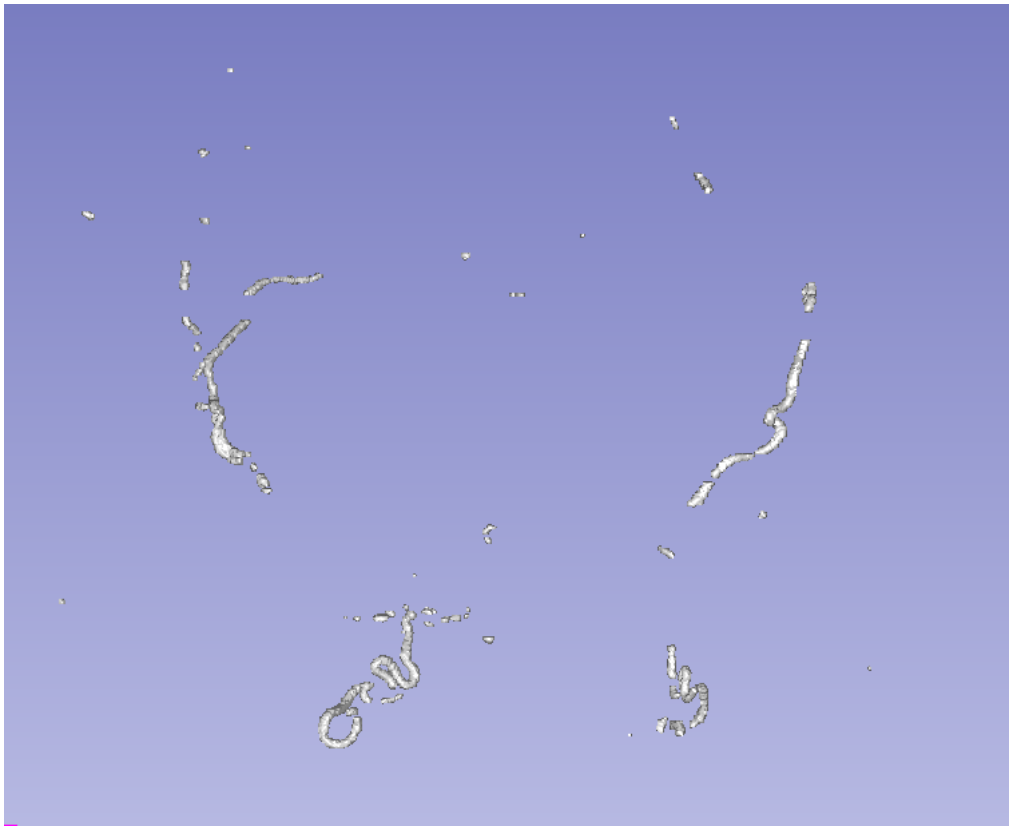
Models with different losses

Two models

Combined results of model from `DiceCELoss` and `Tversky Loss`.

Patient-1

Patient-2



Results show huge improvement in noise regions. Can be further improved with more ensembles.

But the ensemble method is manual right now.

> 💡 `torch.bitwise_and` takes only two inputs. Alternatively, can use simple `&` operator.

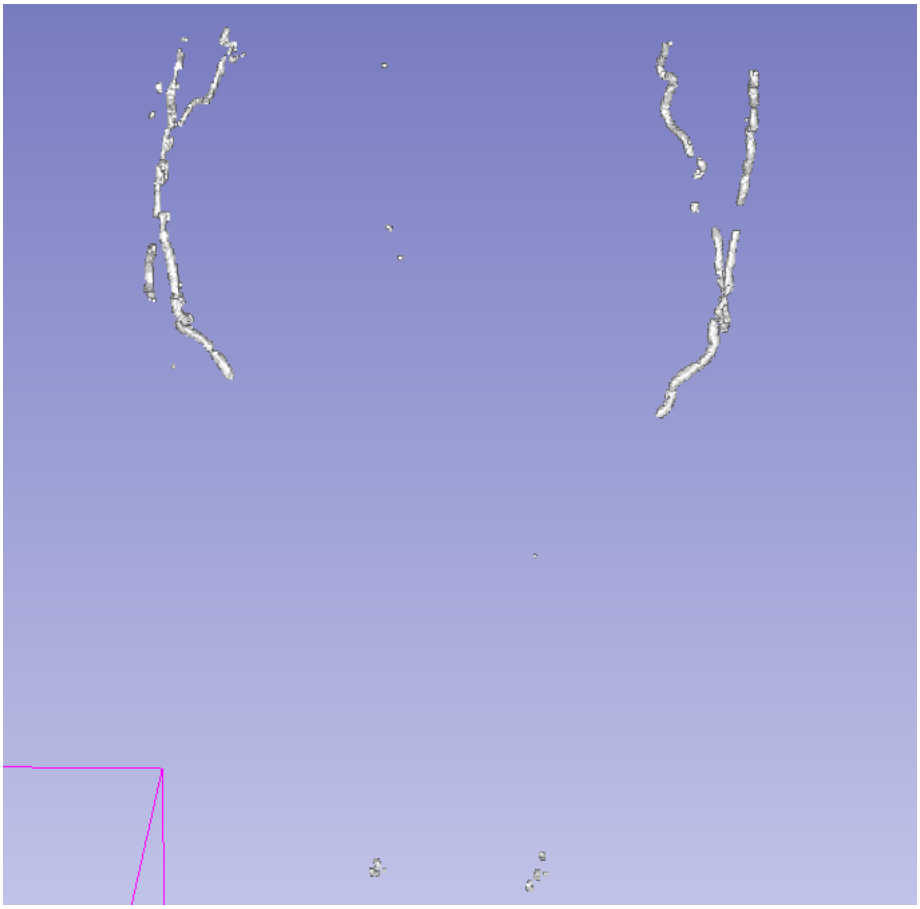
Three models

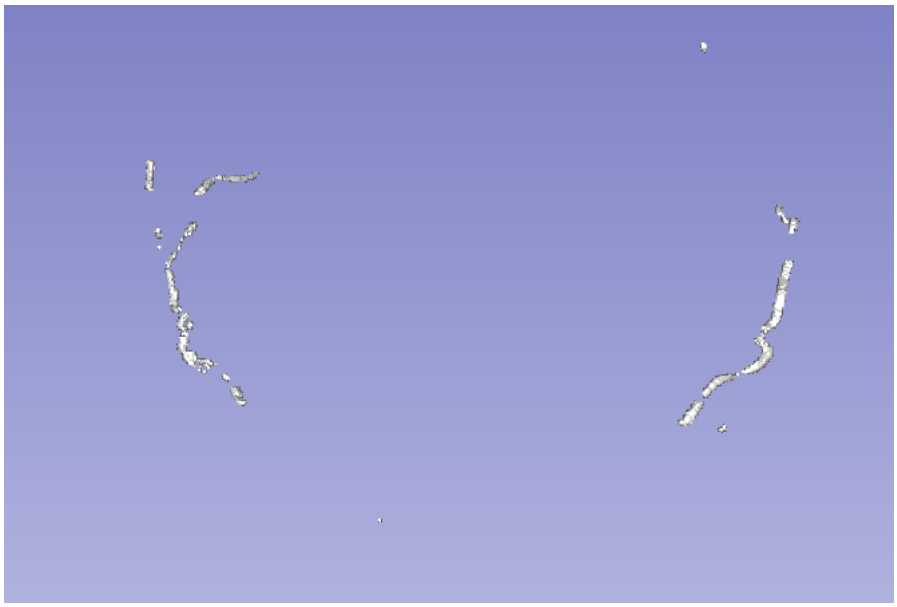
DiceCELoss + GeneralizedDiceCELoss + TverskyLoss

Trained a new model with `GeneralizedDiceCELoss`.

It has a lot of noise during individual testing.

Results improved.

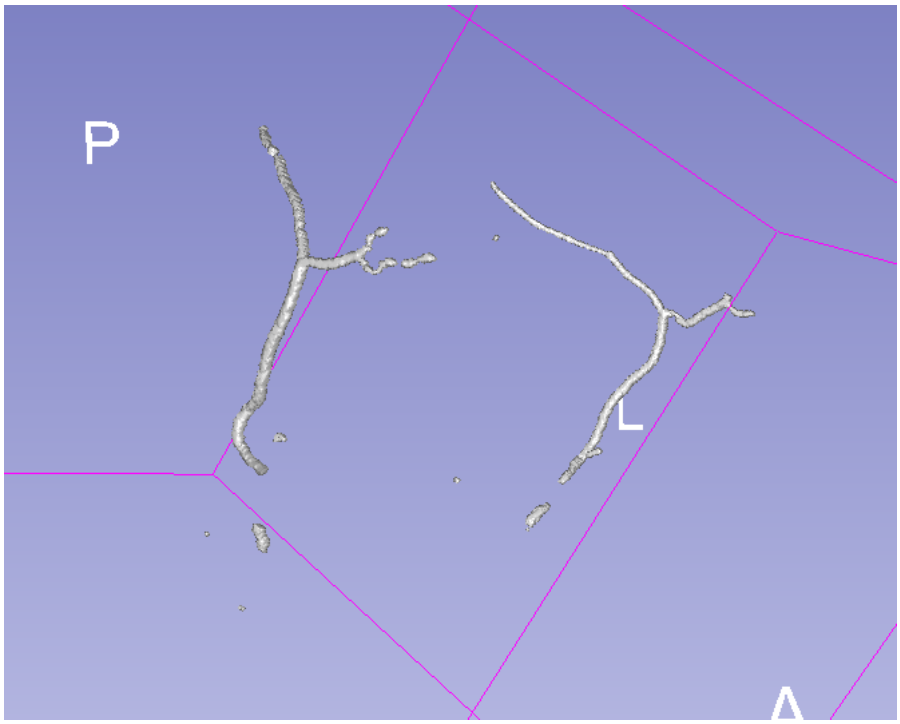




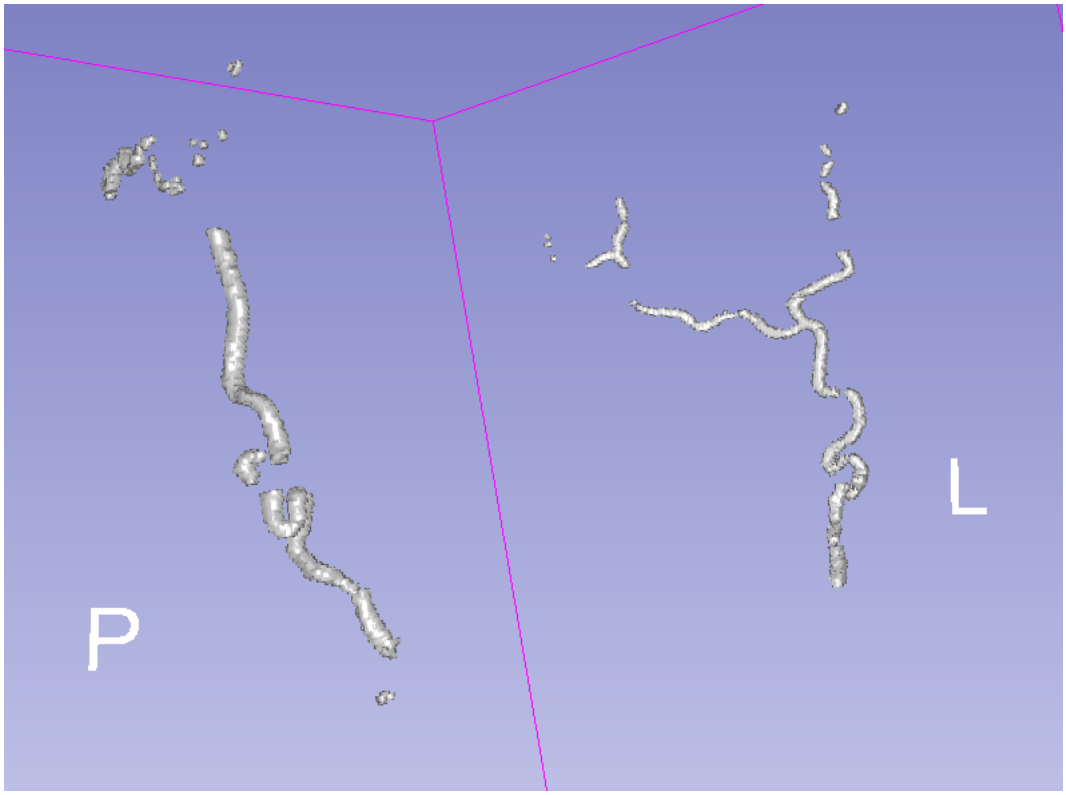
Very less noise now after combining three model predictions. However, facial artery prediction is week. Maybe I need to use **high recall** models.

Testing on validation data

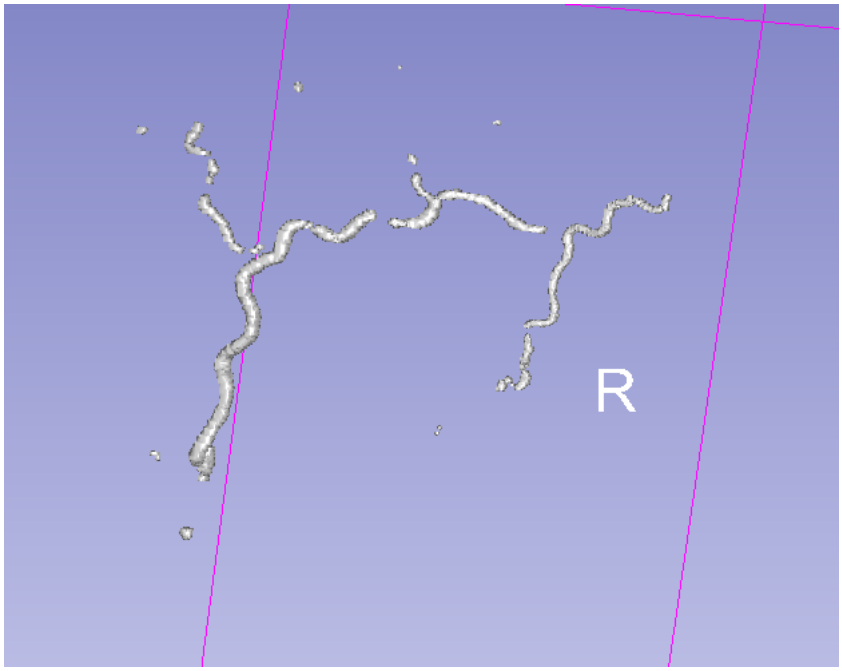
Patient-13



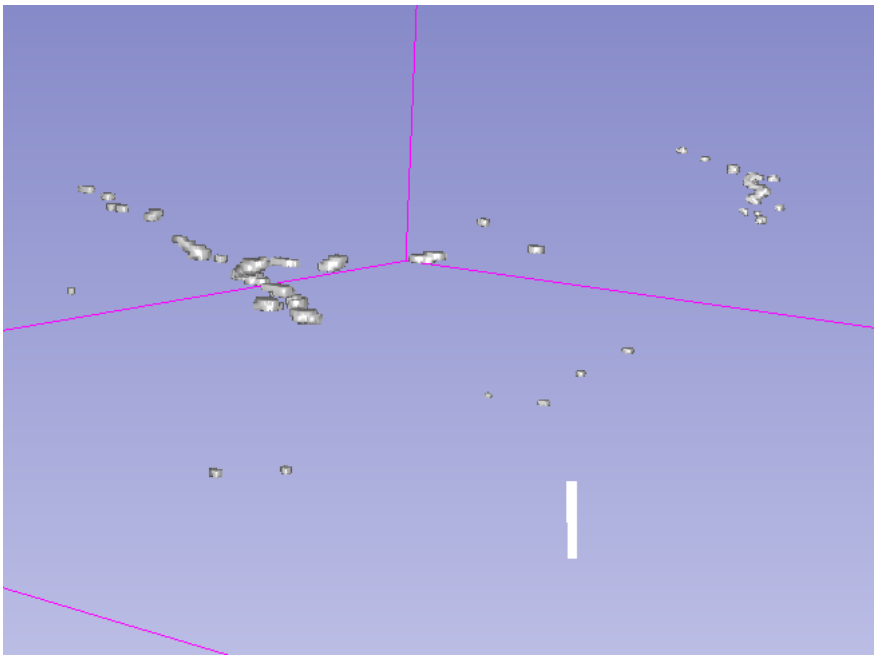
Patient-16



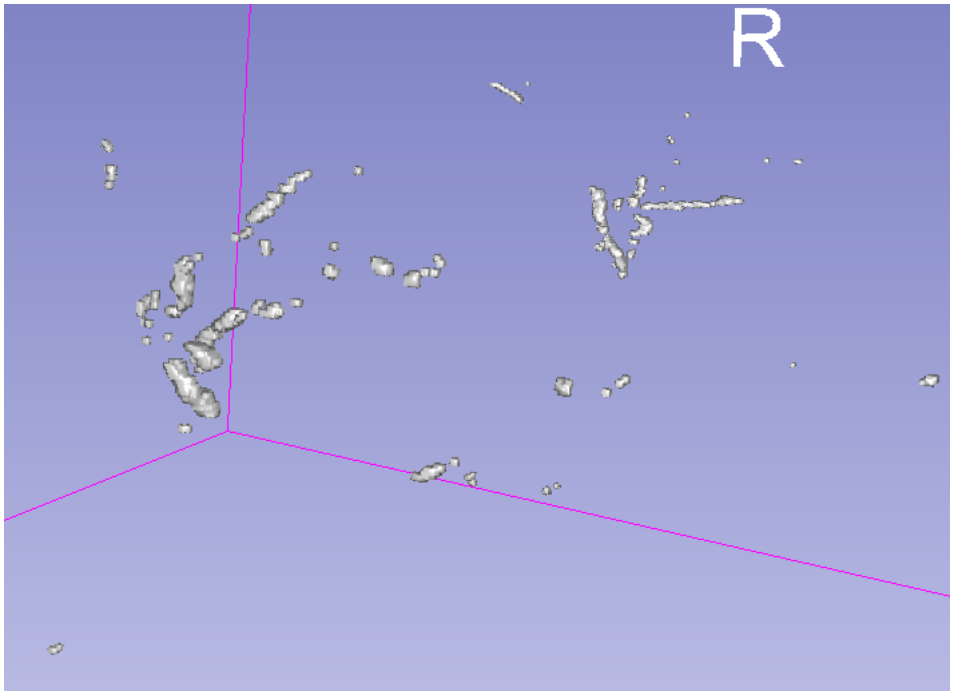
Patient-17



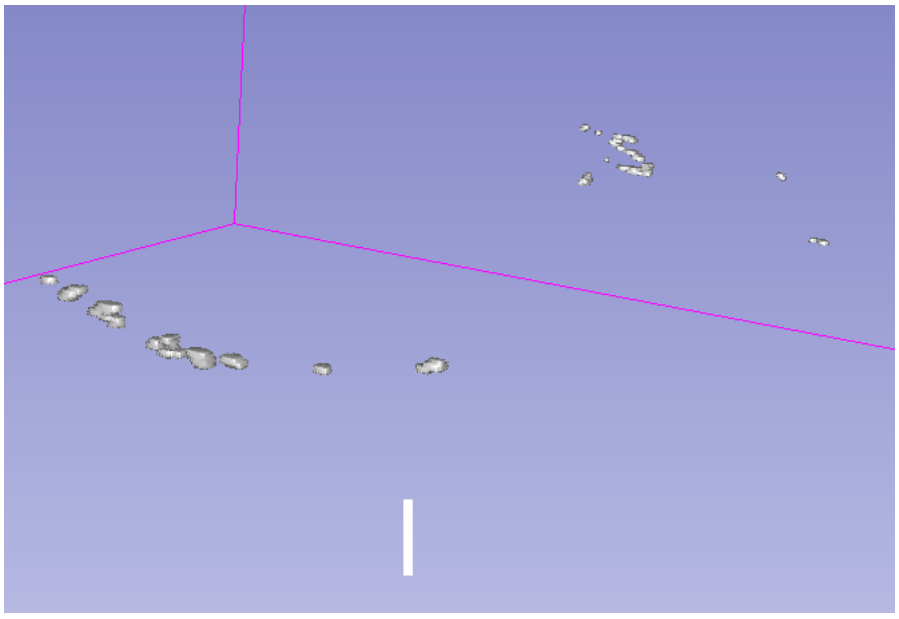
Patient-24



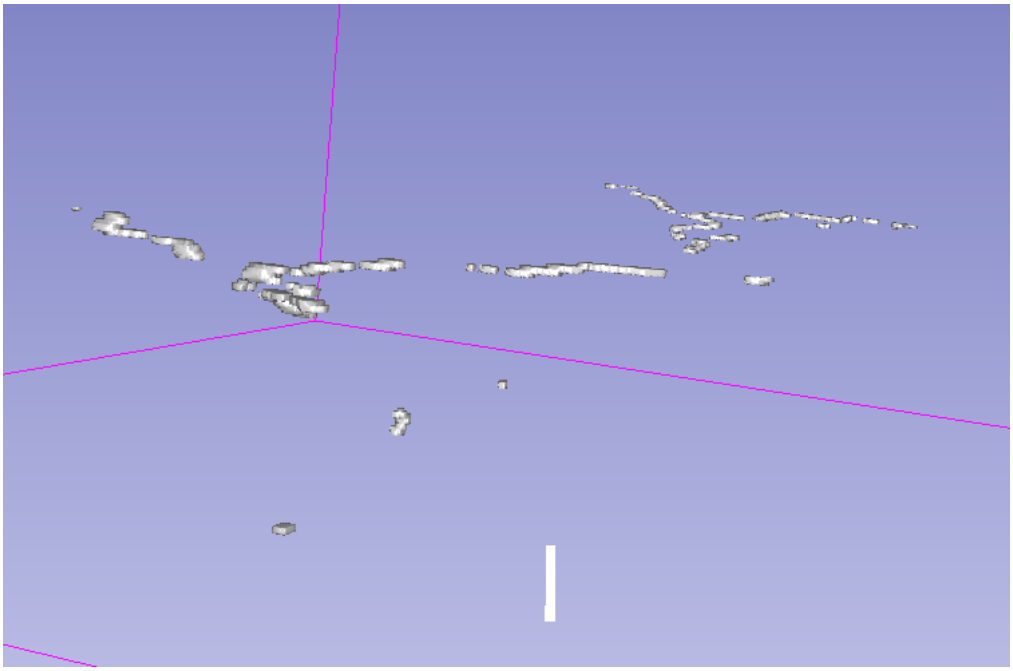
Patient-35



Patient-38

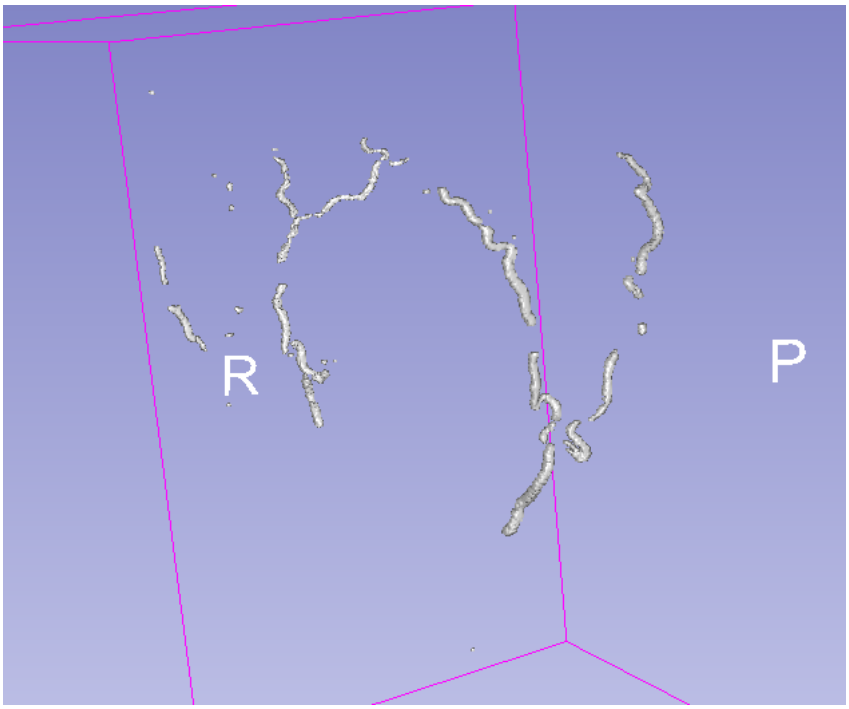


Patient-42

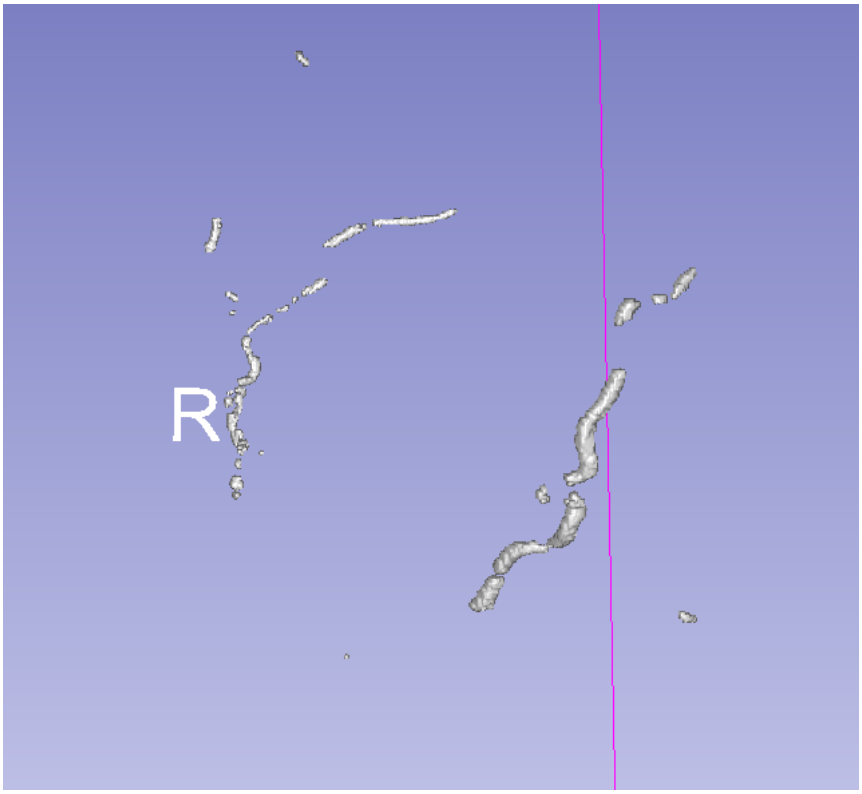


Testing on new test data

Patient-1

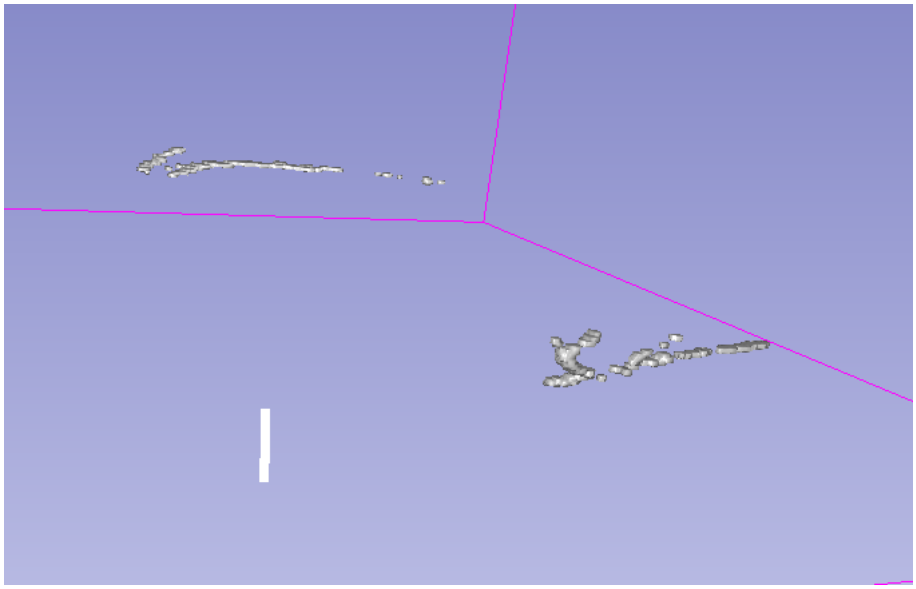


Patient-2

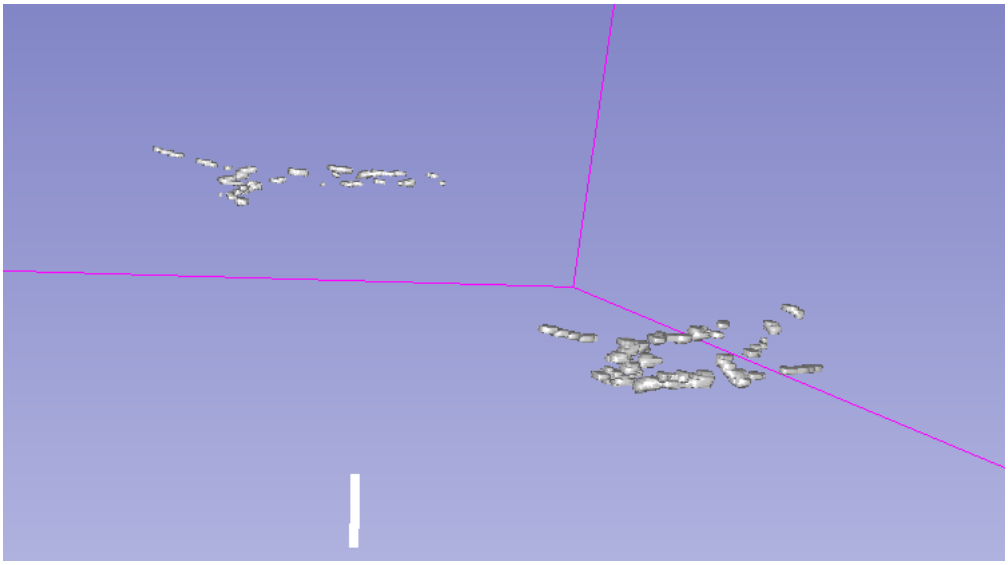


Patient-3

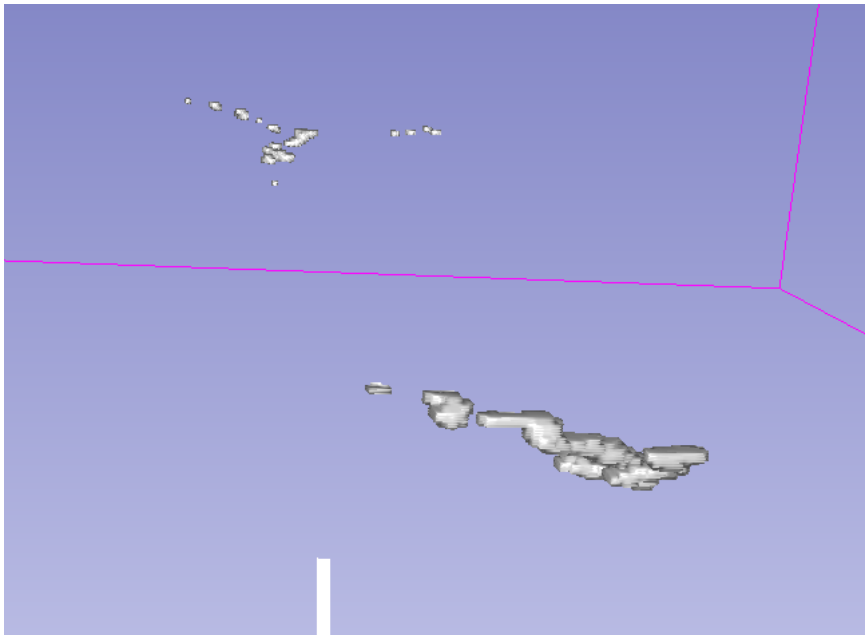
Patient-4



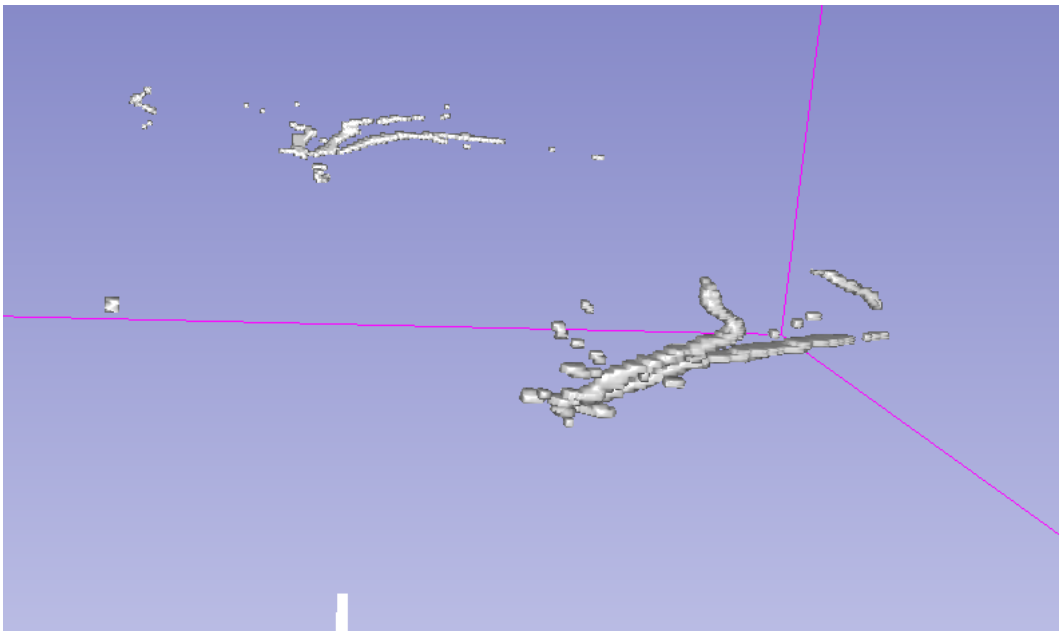
Patient-5



Patient-6



Patient-7



***** Research Phase - IV *****

Idea

Contrast dataset with ensemble learning

Preparing dataset

- Check labels manually

- Fix labels
- Veridy by prinitng shapes

```
Issue files

Extra label: 1, 13
Opposite labels: 2, 3, 4, 8, 9, 10, 14, 19, 24, 29, 30, 31, 32, 34,
```

Todo

Testing on validation data

Testing on new test data

New contrast dataset → ensemble with different losses

Ideas

fine-tuning

Useful Links

MONAI

https://docs.monai.io/en/stable/index.html

https://github.com/Project-MONAI/tutorials/tree/main/3d_segmentation

https://github.com/Project-MONAI/tutorials/blob/main/modules/cross_validation_models_ensemble.ipynb

Annotation

3D Slicer for annotating microscopy data

Resizing 3D Image

Resizing Data - 3D Convolutional Neural Network w/ Kaggle and 3D medical imaging p.4

https://stackoverflow.com/questions/64674612/how-to-resize-a-nifti-nii-gz-medical-image-file

Might be useful

https://www.kaggle.com/code/mechaman/resizing-reshaping-and-resampling-nifti-files/notebook

Best one maybe

https://keras.io/examples/vision/3D_image_classification/

Data loader and training

https://www.youtube.com/watch?v=ScdCQgLtnis&t=829s

https://www.youtube.com/watch?v=PNqnLbzdxwQ

Torch IO

https://torchio.readthedocs.io/_modules/torchio/data/queue.html

Loss calculation

https://github.com/kevinzakka/pytorch-goodies/blob/master/losses.py

https://groups.google.com/g/keras-users/c/U-BHRhy-QQs?pli=1

> https://github.com/pytorch/pytorch/issues/1249

https://discuss.pytorch.org/t/runtimeerror-the-size-of-tensor-a-1966080-must-match-the-size-of-tensor-b-655360-at-non-singleton-dimension-0/38871/5

https://discuss.pytorch.org/t/multiclass-segmentation/54065/44?page=3

https://github.com/wolny/pytorch-3dunet/blob/master/pytorch3dunet/unet3d/losses.py

https://discuss.pytorch.org/t/how-to-select-a-loss-function-for-3d-segmentation-networks/2095

Tversky Loss

https://arxiv.org/abs/1706.05721

Precision-Recall

https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#:~:text=A system with high precision,with all results labeled correctly.

https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

Papers

Coronary artery segmentation under class imbalance using a U-Net based architecture on computed tomography angiography images

https://www.nature.com/articles/s41598-021-93889-z

Ensembling Low Precision Models for Binary Biomedical Image Segmentation

https://openaccess.thecvf.com/content/WACV2021/papers/Ma_Ensembling_Low_Precision_Models_for_Binary_Biomedical_Im