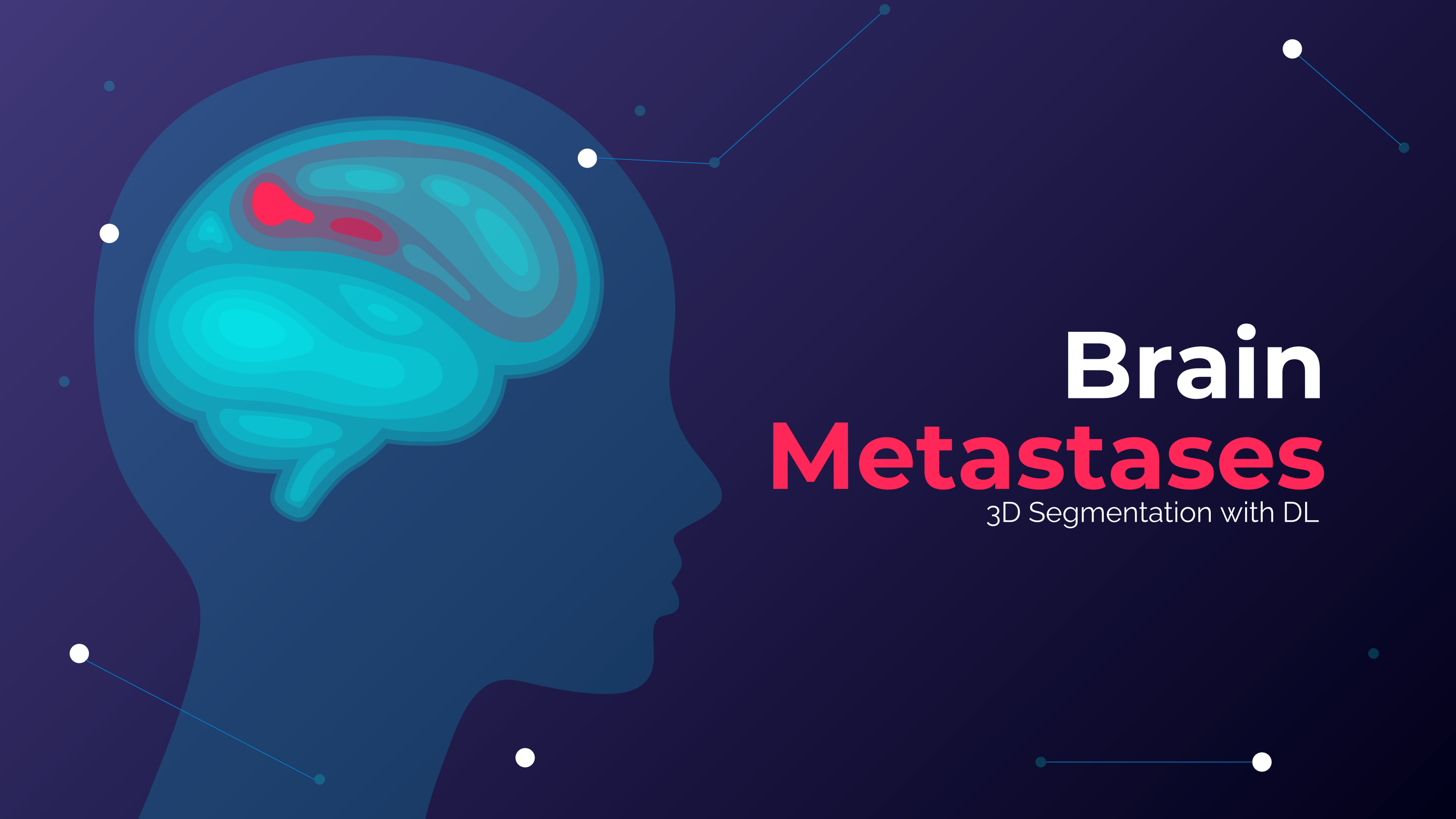


Brain Metastases

3D Segmentation with DL

Available Datasets

Dataset	N° of patients	Access Status	T1 post-contrast	T1 pre-contrast	T2 FLAIR	T2	Enhanced Tumor (ET)	Non Enhancing Tumor Core (NETC)	Edema (SNFH)	Necrotic portions	Sub Centimeter Lesions	Raw DICOM and NifTi formatted files	Demographics	Clinical Data
Brain metastases MRI dataset	75	Open	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes
BrainMetShare	156	Open	Yes	Yes	Yes	No	Yes	No	No	No	No	Yes	No	No
Pretreat MetsToBrain Masks	200	Open (Aspera Issue)	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
UCSF-BMSR	412	Open	Yes	Yes	Yes	No	Yes	No	No	No	No	Yes	Yes	No
BraTS 2023 Challenge Data	412	Participation Required	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No

Available Datasets

Public dataset	Data publisher	Number of cases	Difference from BraTS datasets
NYUMets[2]	New York University (NYU)	1,429 patients	• Contains post therapy cases • Not all patients have images • Many cases without brain metastasis
BrainMetShare[4] [5]	Stanford University	156 patients	• Not containing T2 sequence • Contains post therapy cases • Available in JPEG format
UCSF-BMSR[26]	University of California San Fransico (UCSF)	412 patients	• Not containing T2 sequence • Contains post therapy cases
Brain-TR-GammaKnife[46]	University of Mississippi (UMMC)	47 patients	• Recently published
MOLAB[47]	University of Castilla-La Mancha	75 patients	• Contains post therapy cases • Recently published

Annotations Explained

Annotation Type	Purpose	Importance for 3D-Segmentation
T1 pre-contrast (t1n)	Provides a baseline image	Helps in comparing with post-contrast images to detect abnormalities.
T1 post-contrast (T1c)	visibility of lesions (highlight disruptions in the blood-brain barrier)	Critical for delineating tumor boundaries and differentiating tumor from normal brain tissue.
T2 FLAIR (t2f)	visibility of lesions, edema, or gliosis (suppresses fluid signals)	Essential for identifying and segmenting peritumoral edema and differentiating it from other tissues.
T2 (t2w)	visualize fluid content and edema (high contrast images of brain tissues)	Helps in visualizing the entire extent of the tumor environment.
Enhanced Tumor (ET)	Indicate active tumor.	Critical for detailed tumor characterization in 3D.
Non-enhancing Tumor Core (NETC)	Central non-enhancing part of the tumor, often necrotic or cystic.	Important for assessing the internal structure of the tumor in 3D.
Edema (SNFH)	edema and infiltrative tumor tissue (abnormal signal on FLAIR images)	Essential for accurate segmentation of tumor-affected regions.
Necrotic Portions	areas of tumor that have necrosis.	Important for assessing tumor aggressiveness and predicting treatment response.
Sub-centimeter Lesions	Small lesions that are challenging to detect and segment.	Enhances the sensitivity of segmentation algorithms in detecting very small tumors.
Raw DICOM and NifTi files	Usually NifTis are used in research	Essential for using clinical imaging data in segmentation algorithms.
Demographics	Includes patient age, sex, health history.	Useful in correlating demographic features with imaging findings.
Clinical Data	Includes detailed health records and clinical history.	Critical for correlating clinical features with imaging findings .

Choice of Datasets?

Compatibility

Ensuring compatibility and uniformity across segments and labels of different datasets is resource-intensive and challenging

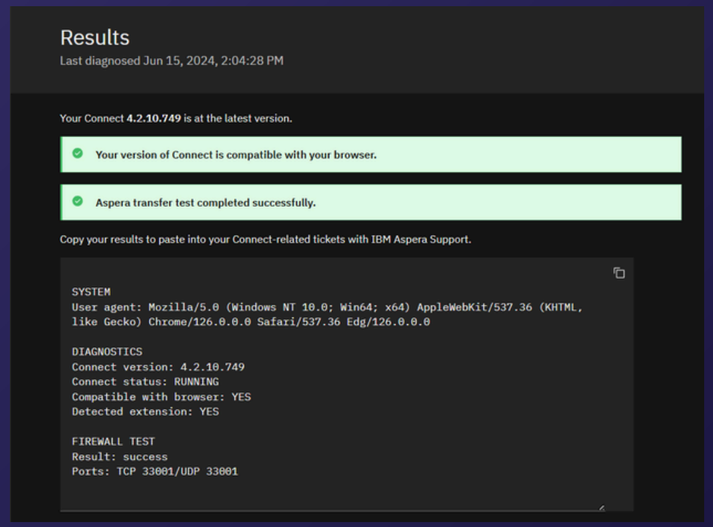
data use agreements

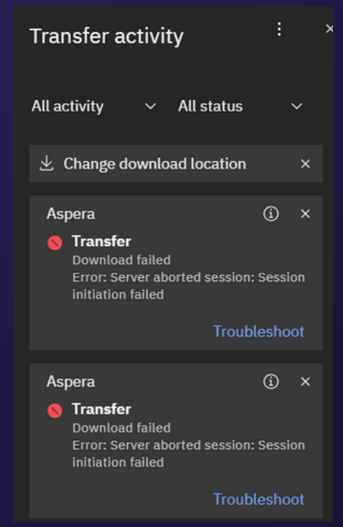
Analysing generalization and publishing an article possibility depends on the use agreements.

Issues and Alternatives

Aspera Connect Issue

Although **Pretreat MetsToBrain Masks** is the most annotations intensive, **there's still an issue with Aspera**.





Alternatives

I participated in **Brats Met 2023 and 2024** and **aquired both datasets**.

What follows is a comparison between these two and a discussion as to why they are preferred.

Brats Met

BraTS MET 2023

The BraTS METS 2023 dataset features a robust selection of cases from prestigious institutions, all of which underwent rigorous annotation and vetting processes. Contributors include:

Table 2: Dataset sources in the BraTS-METS 2023 challenge. In the training dataset, 474 cases from UCSF and Stanford were included as optional because they did not have original T2 weighted images.

Dataset Source	Total cases reviewed	Excluded	Training	Validation	Test
Duke	37	0	26	4	7
CairoU	45	10	32	1	2
Missouri	25	3	16	2	4
WashU	40	1	27	4	8
Yale	225	30	137	20	38
NYU*	221	57	164	0	0
UCSF^	560	236	324	0	0
Stanford^	150	0	150	0	0
Total	1,303	337	402 (474 optional)	31	59

Table 3: Lesion count and sizes for each dataset group.

Dataset Group	ET lesion-count (total)	ET lesion-count median (IQR)	ET lesion-size median (IQR)	WT lesion-count (total)	WT lesion-count median (IQR)	WT lesion-size median (IQR)
Training* ($n = 402$)	3076	3 (7)	65 (287)	2618	3 (5)	121 (804)
Validation ($n = 31$)	139	3 (4)	141 (664)	119	3(3)	591 (3318)
Testing ($n = 59$)	218	2 (3)	132 (613)	193	2 (3)	322 (8624)

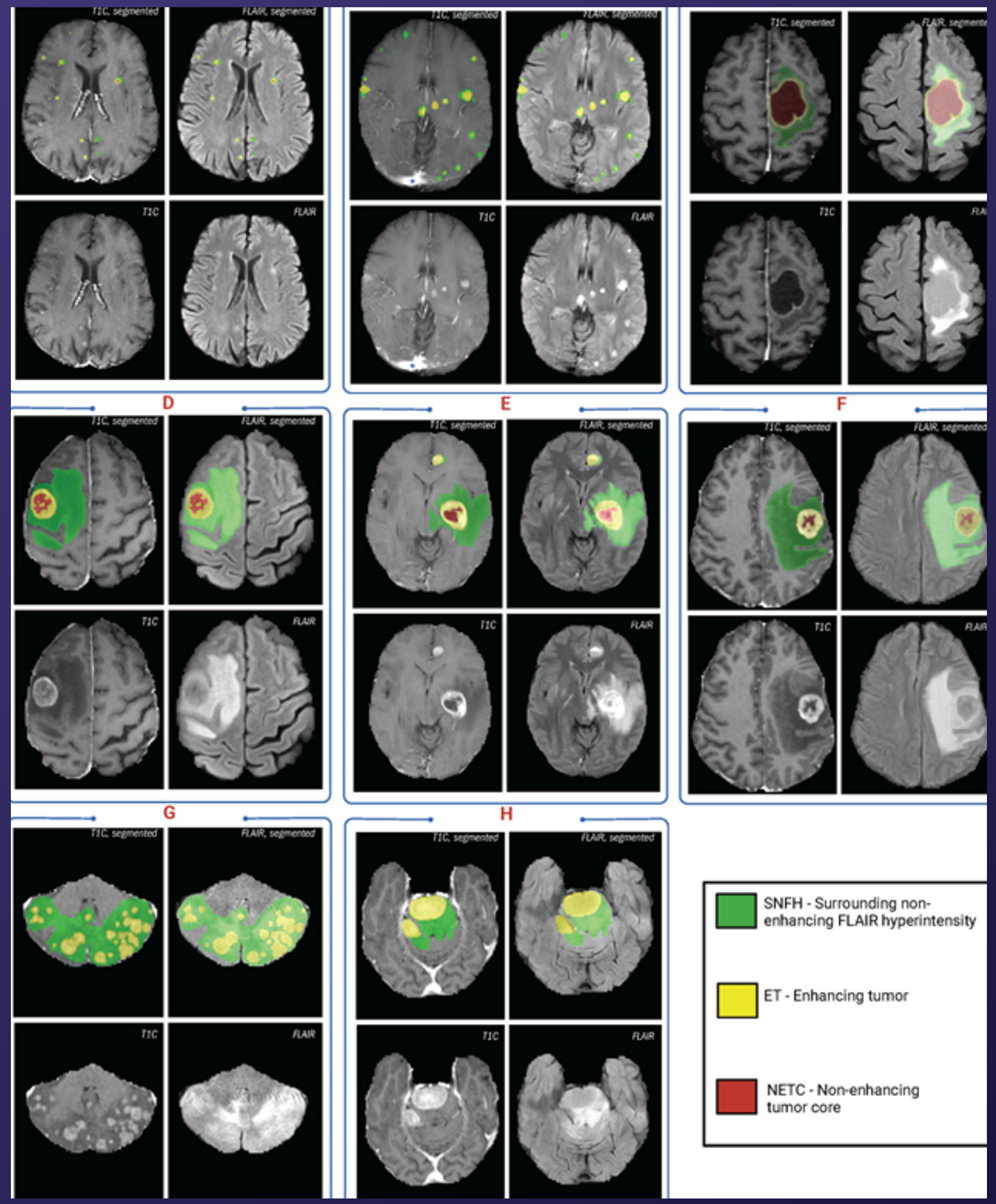
* The training group does not include the optional UCSF and Stanford datasets.

BraTS MET 2024

The BraTS METS 2024 dataset builds on the 2023 dataset by including post-treatment changes and other instances. It offers a comprehensive collection of pre-treatment brain metastases 4 5 multi-parametric magnetic resonance imaging (mpMRI) scans, ensuring data uniformity crucial for developing and validating advanced machine learning models. New external sources include:

- **164 NYU cases** with all 4 sequences in BraTS format, requested directly from NYU (data agreement reviewed and confirmed to be compatible with our research use).
- **UCSF-BMSR MRI Dataset** with full BraTS format annotations on 324 cases, available as Training Dataset 2. Additional cases lacking the full format are not utilized due to compatibility requirements.

Brats Mets 2023



The annotation of tumor sub-regions aligned with Visually AcceSAble Rembrandt Images (VASARI) feature visibility and encompassed three labels: Gd-enhancing tumor (ET - label 3), surrounding non-enhancing FLAIR hyperintensity (SNFH - label 2), and the non-enhancing tumor core (NETC – label 1). ET is described as the enhancing portion of the tumor, characterized by areas of hyperintensity in T1Gd that are brighter than T1. NETC is identified as the presumed necrotic core of the tumor, which is evident as a non-enhancing focus surrounded by enhancing tumor. SNFH is defined as the peritumoral edema and tumor infiltrated tissue, indicated by the abnormal hyperintense signal on the T2-FLAIR images, which includes the infiltrative

1. Whole Tumor (WT) = Label 1 + Label 2 + Label 3
2. Tumor Core (TC) = Label 1 + Label 3
3. Enhancing Tumor (ET) = Label 3

Our annotation and approval pipeline, as previously described, was applied to datasets from a variety of institutions, including New York University (NYU), Yale University, Washington University, Cairo University (CairoU), Duke University, and the University of Missouri. The annotated NYU dataset is uniquely hosted on the NYU website[2], separate from the public BraTS repository. As for the UCSF dataset, synthetic T2 images were generated and shared on the UCSF website[3]. The Stanford University dataset, despite being publicly available, was not incorporated into our primary dataset due to the lack of T2 image sequences. These datasets were available and optional for additional training. For logistical reasons, the UCSF, Stanford, and NYU datasets were excluded from the validation and test phases of our project.

$$\text{Lesion-wise Dice Score} = \frac{\sum_i^L Dice(l_i)}{TP + FN + FP} \quad (1)$$

$$\text{Lesion-wise HD95} = \frac{\sum_i^L HD_{95}(l_i)}{TP + FN + FP} \quad (2)$$

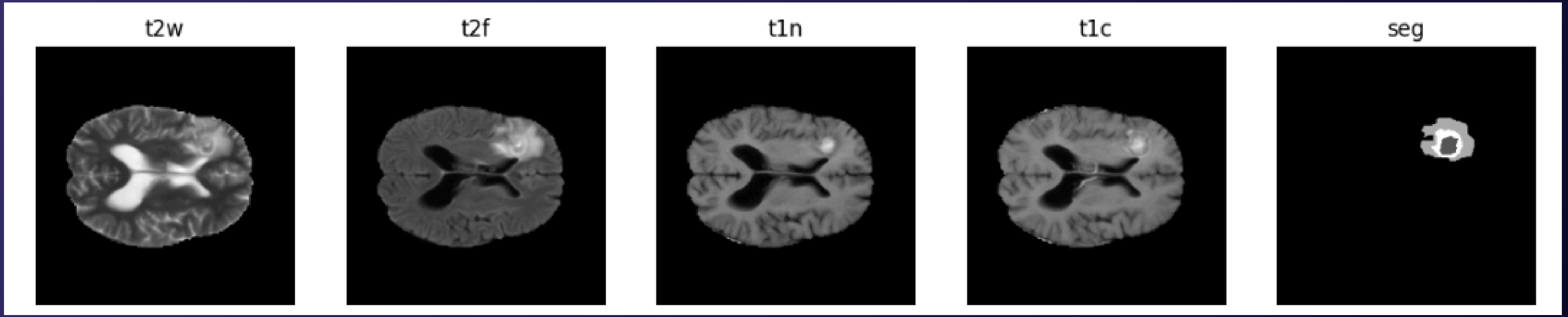
where L is the total number of GT lesions and TP, FP, FN are the number of true positive, false positive and false negative lesions respectively.

metrics: lesion-wise Dice and lesion-wise HD95. These metrics have been developed primarily to evaluate the performance of models at the level of individual lesions, rather than on a whole-image basis. This approach ensured that our evaluation did not favor models that only captured large lesions, a limitation commonly observed with standard Dice scores. By assessing models on a lesion-by-lesion basis, we gained insights into their ability to segment all sizes of BMs accurately.

The mpMRI scans included four sequences: non-enhanced T1, post-gadolinium-contrast T1 (T1Gd), T2, and non-enhanced T2-FLAIR, procured from various scanners and protocols. Standardized pre-processing was applied to all the BraTS-METS mpMRI scans. Specifically, the applied pre-processing routines included conversion of the DICOM files to the NIfTI file format, co-registration to the same anatomical template (SRI24)(Rohlfing et al., 2010), resampling to a uniform isotropic resolution ($1mm^3$), and, finally, skull stripping (Isensee et al., 2019). The pre-processing pipeline was made publicly available through the Cancer Imaging Phenomics Toolkit (CaPTk) (Pati et al., 2020; Rathore et al., 2018) and the Federated Tumor Segmenta-

Multimodal Brain Tumor Image Segmentation = Brats

Four 'channels' of information - 4 different volumes of the same region
i. Native (T1)
ii. Post-contrast T1-weighted (T1CE)
iii. T2-weighted (T2)
iv. T2 Fluid Attenuated Inversion Recovery (FLAIR) volumes

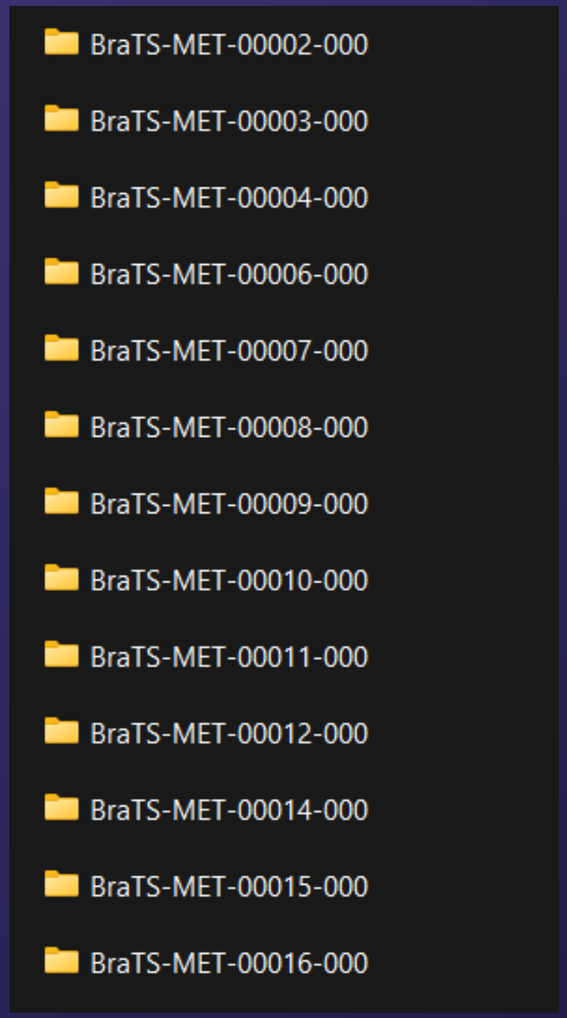


Annotations

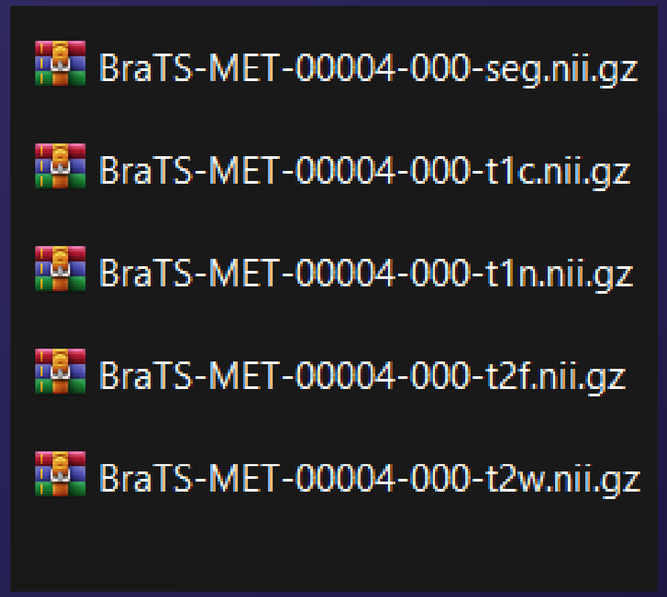
- **Label 1:** The nonenhancing tumor core (NETC – label 1)
- **Label 2:** The peritumoral edematous/infiltrated tissue: Surrounding non-enhancing FLAIR hyperintensity (SNFH) - label 2
- **Label 3:** Enhancing tumor (ET) - label 3

Training set:

234 folders
161+ 73

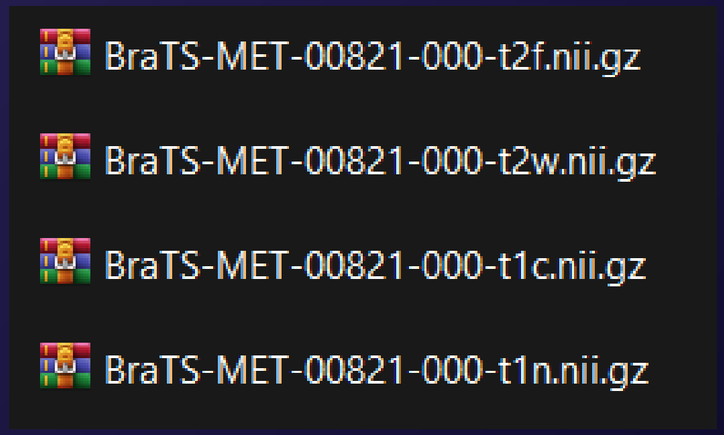


Contents of each folder



Validation set:

31 Folders



Compliance with data use agreement

Evaluating the model's ability to generalize is a crucial aspect of research, particularly in the medical field where it has garnered significant attention. This evaluation typically involves using train/test splits from the initial dataset and subsequently testing the model on additional datasets. **The integration of BraTS datasets with other data sources for such research purposes is permissible, subject to specific conditions as outlined in the respective data use agreements**

Explicit Documentation:

This involves the use of medications, surgery, radiation therapy, chemotherapy or other medical procedures to treat a particular illness

Citing and Acknowledgments:

Behavioral therapies involve various techniques that are used to treat mental health conditions, such as depression, anxiety and addiction

Compliance with Embargo and Publication Policies:

Adhere to **embargo periods** and publication restrictions specified by the BraTS organizers.

- For **the 2024 dataset** (only training datasets available for now), this involves **waiting until the official release of the joint overview paper** before publishing comprehensive challenge results.
- **The 2023 dataset** is already available for immediate academic dissemination.

Analysis of Acquired BraTS Datasets and Research Metrics from the Brats 2023 Challenge

1st NVAUTO (SegResNet from MONAI Auto3DSeg)

- MONAI native (uses transforms, loaders, losses, networks components of MONAI)
- 4-channel input, which is a concatenation of four different MRI scans
- Input data is normalized to have zero mean and unit standard deviation for each channel.
- Employs random cropping to a fixed size of 224x224x144 pixels
- AdamW optimizer with a learning rate of 2e-4 is used in combination with a cosine annealing scheduler
- Model is trained for a range of 300 to 1000 epochs, using 5-fold cross-validation
- A combined Dice-Focal loss function is utilized for training
- Data augmentation techniques include spatial transformations (random rotations, scaling, flips) and intensity modifications (random adjustments to intensity/contrast, addition of noise, and blur)
- Code reference: GitHub - MONAI and SegResNetDS

2nd SY (3D TransUNet Model (Chen et al., 2023a))

- 3D nnUNet as the CNN Encoder + Decoder
- 12-layer ViT as the Transformer Encoder with ImageNet pretrained weights
- A hybrid loss function consisting of pixel-wise cross entropy loss and dice loss
- Pre-train the transformer blocks using Masked Autoencoder (He et al., 2022)
- Code reference: 3D TransUNet Model

3rd blackbean (STU-Net)

- A scalable and transferable version of nnUNet
- Larger input patch size: 160 x 160 x 160
- Poly decay policy
- Code reference: STU-NET and nnUNetV1

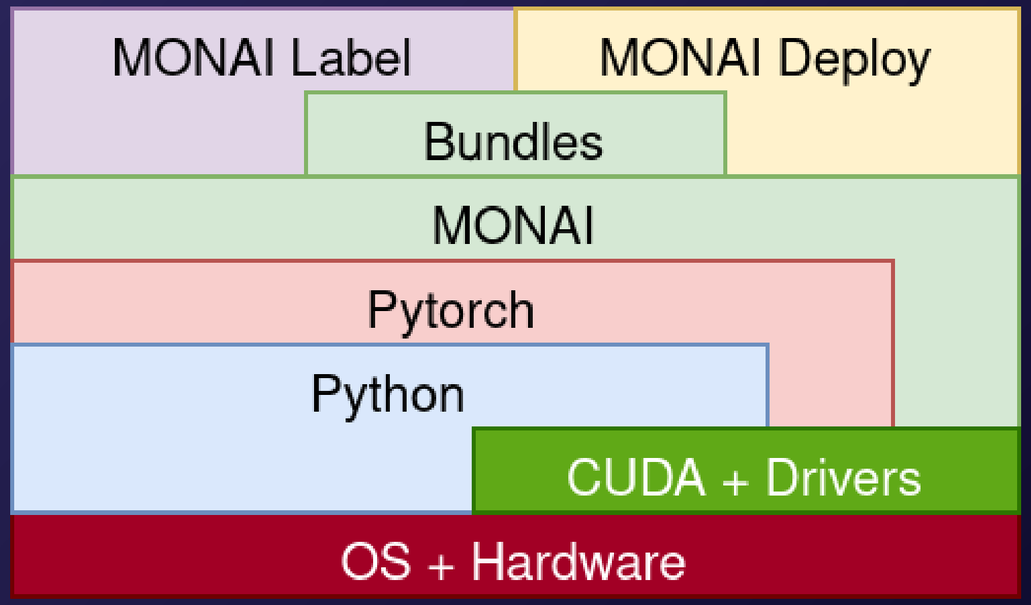
4th CNMCPMI2023 (Label-wise model ensemble approach)

- nnU-Net and Swin UNETR CNN + ViT
- Outputs of these networks are then subjected to a non-linear function
- Processed outputs are combined through model ensembling to create ensembled predictions
- Label-wise post-processing is then applied to these ensembled predictions to produce the final predictions for each label

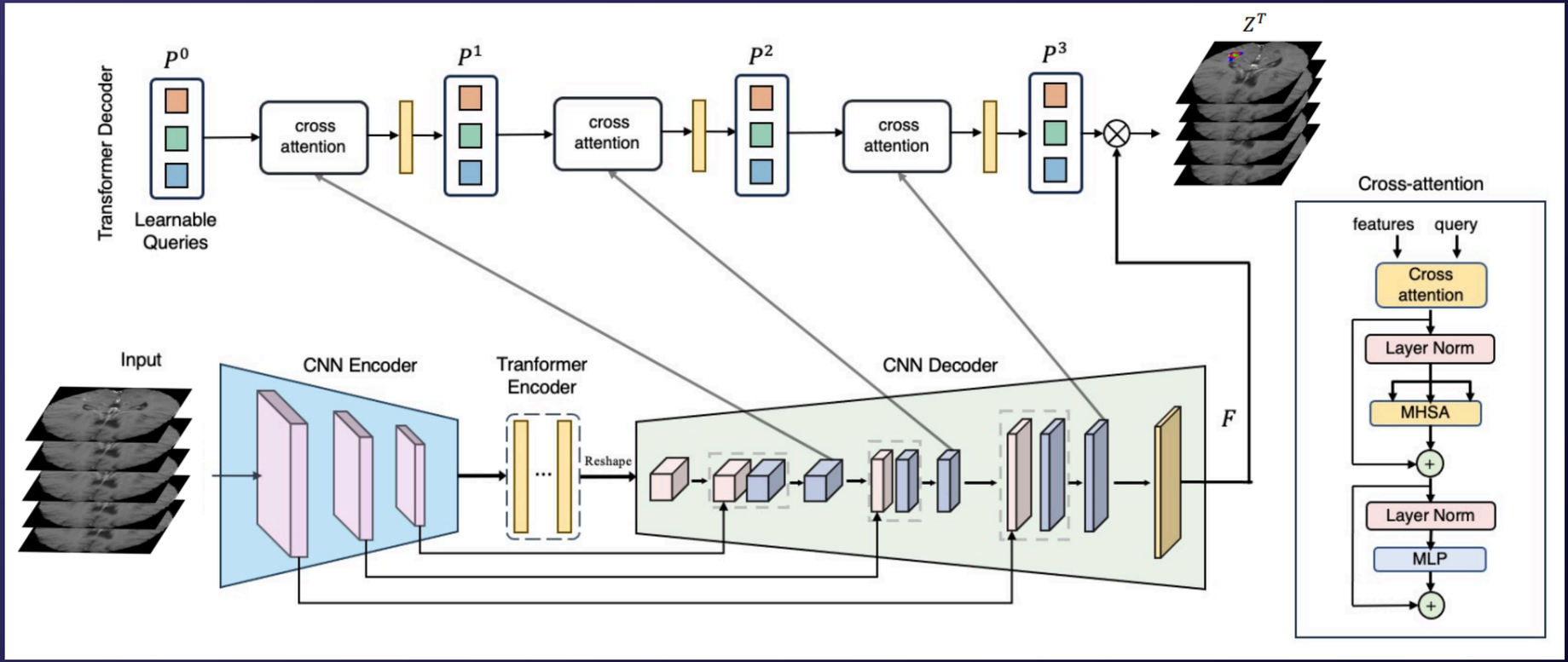
Team Name	ET		TC		WT	
	Dice score	Rank	Dice score	Rank	Dice score	Rank
NVAUTO	0.60 ± 0.24 (0.58)	1	0.65 ± 0.25 (0.60)	1	0.62 ± 0.24 (0.61)	1
SY	0.57 ± 0.28 (0.57)	2	0.62 ± 0.29 (0.64)	2	0.60 ± 0.29 (0.61)	2
blackbean	0.57 ± 0.26 (0.58)	2	0.61 ± 0.28 (0.58)	3	0.57 ± 0.28 (0.57)	4
CNMCPMI2023	0.55 ± 0.28 (0.64)	4	0.60 ± 0.30 (0.69)	4	0.58 ± 0.29 (0.64)	3

Medical Open Network for AI : MONAI is a PyTorch-based, open-source framework for deep learning in healthcare imaging, part of the PyTorch Ecosystem.
Its ambitions are:

- developing a community of academic, industrial and clinical researchers collaborating on a common foundation;
- creating state-of-the-art, end-to-end training workflows for healthcare imaging;
- providing researchers with an optimized and standardized way to create and evaluate deep learning models.

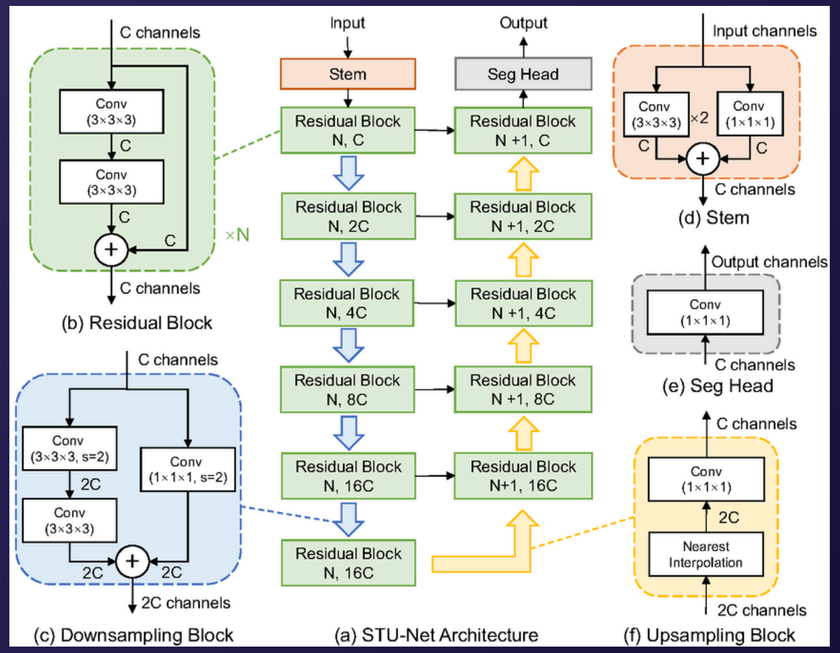


3D TransUnet



STU-Net

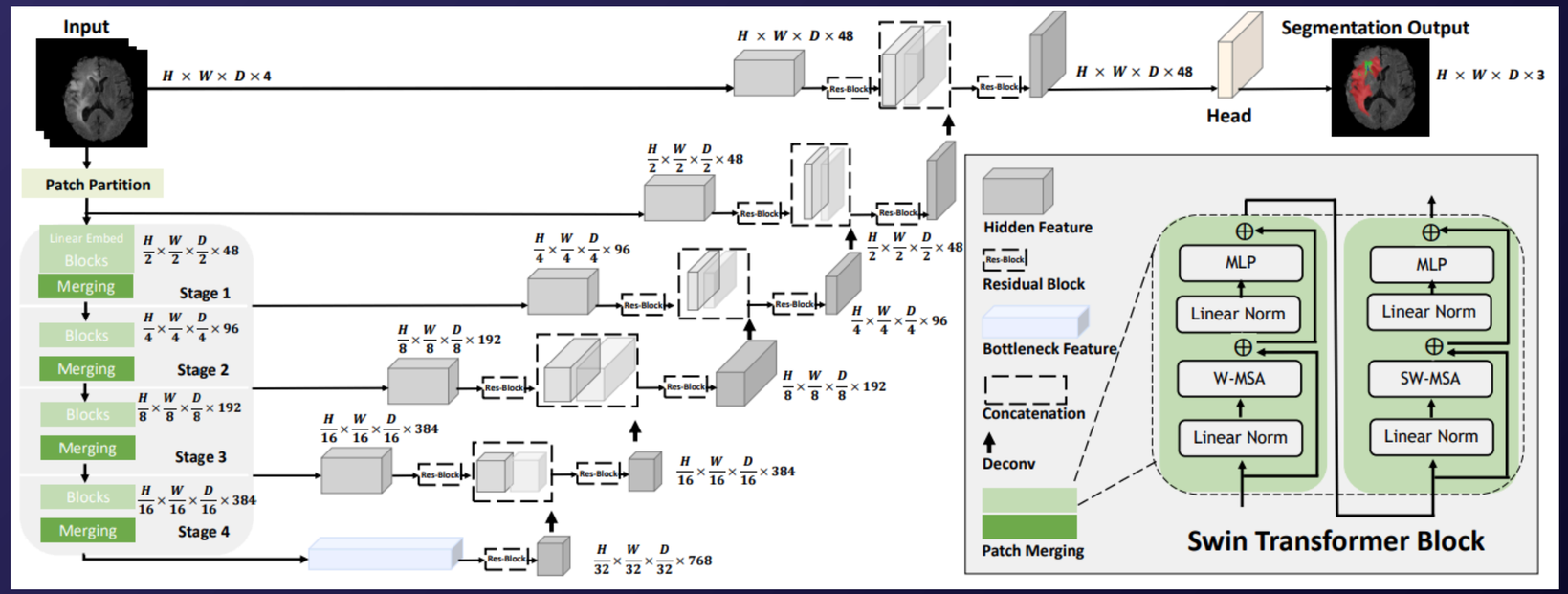
STU-Net is a scalable medical image segmentation model built on the robust **nnU-Net** framework. It offers different model sizes to suit various computational needs, with its largest variant, STU-Net-H, featuring 1.4 billion parameters. Pre-trained on the **TotalSegmentator dataset** with over 100,000 annotations, STU-Net is highly transferable, allowing effective fine-tuning for a wide range of medical imaging tasks



The **nnU-Net**, which stands for "no-new-Net," builds upon the U-Net architecture by creating a **robust and self-adapting framework that leverages both 2D and 3D configurations** of the classic U-Net. This approach is designed to streamline the process of applying neural networks to medical image segmentation by reducing the need for custom adaptations and manual tuning for different datasets.

Swin UNETR

Swin UNETR (Swin Transformers for UNet-based Transformer) leverages the strengths of **FCNNs** and combines them with the capabilities of transformers, particularly suited for handling long-range dependencies in data. Traditional FCNNs, despite their effectiveness in medical image segmentation, often struggle with capturing **long-range information** due to the limited kernel size of convolution layers. This can lead to challenges in accurately segmenting tumors that **vary in size**. To overcome this, Swin UNETR utilizes a **hierarchical Swin transformer** as an encoder to process multi-modal MRI data, which is reformulated into a 1D sequence of embeddings.



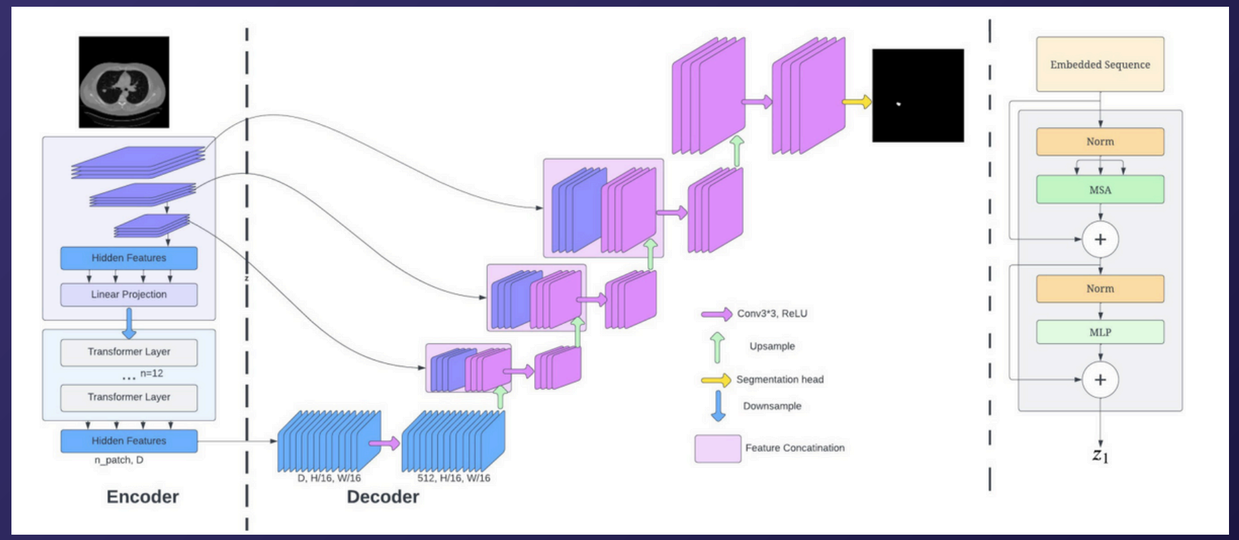
3D TransUnet

VS

Swin UNETR

TransUNet combines a Transformer encoder with a convolutional decoder, leveraging **the global receptive field of transformers alongside the local processing strengths of CNNs.**

Swin UNETR uses a **fully transformer-based approach**, employing Swin Transformers in both the encoder and decoder to manage **hierarchical features and long-range interactions** more effectively.



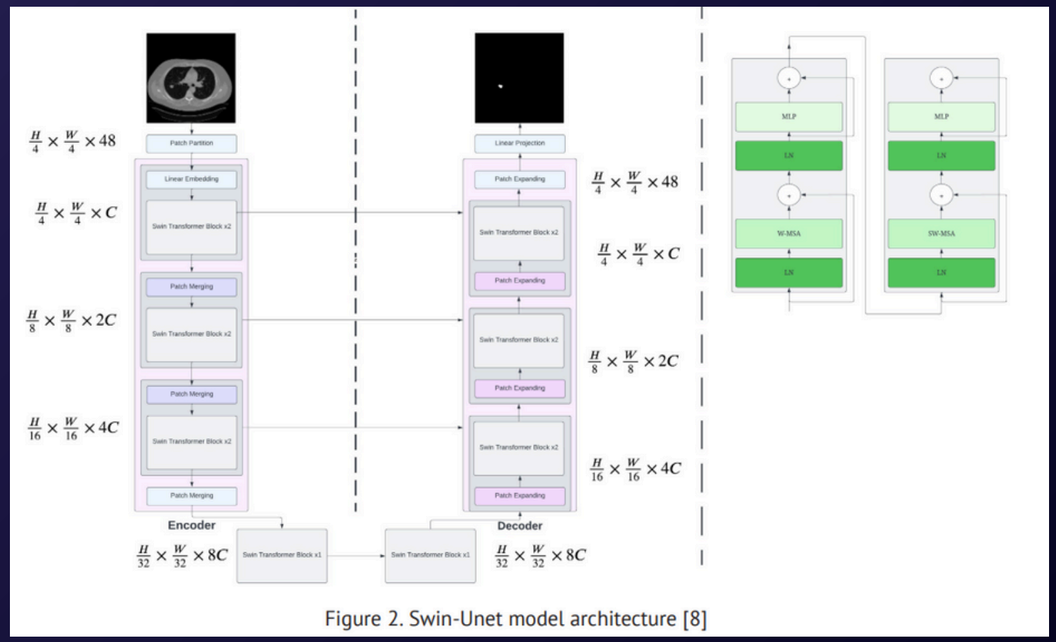
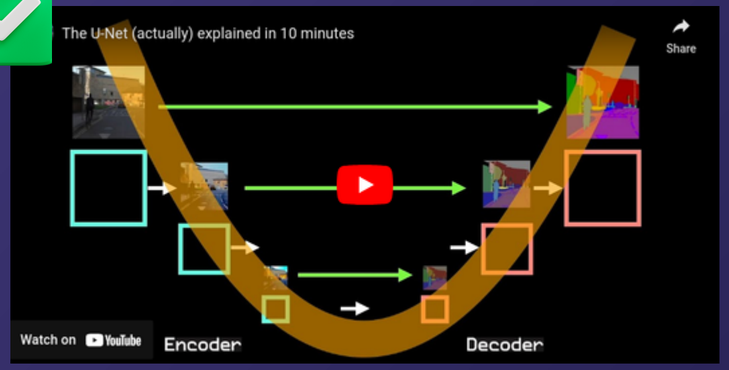
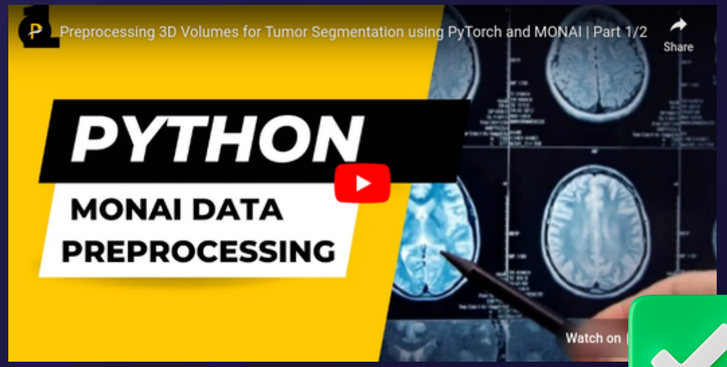
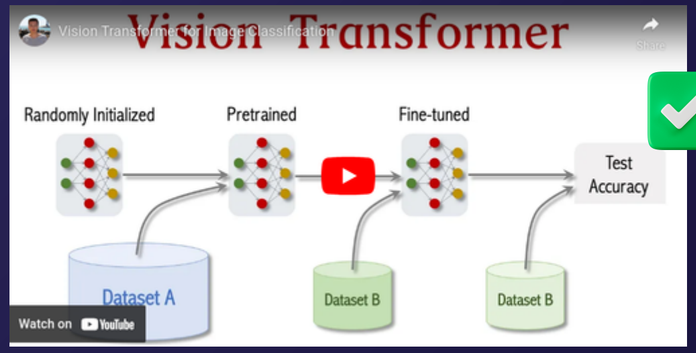


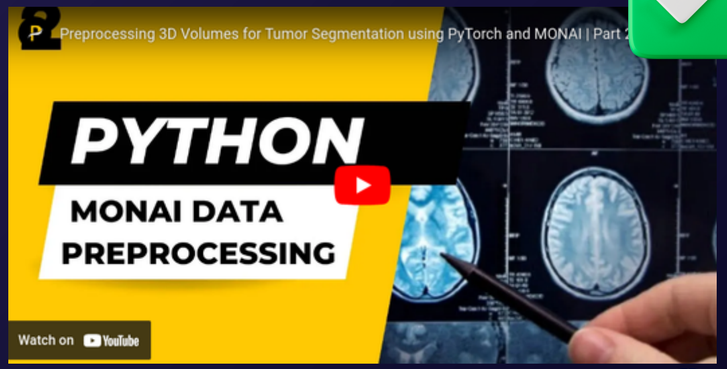
Figure 2. Swin-Unet model architecture [8]

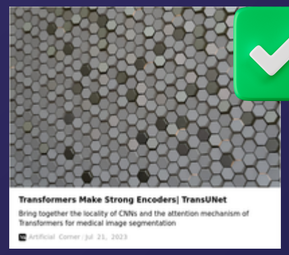
Watched Tutorials

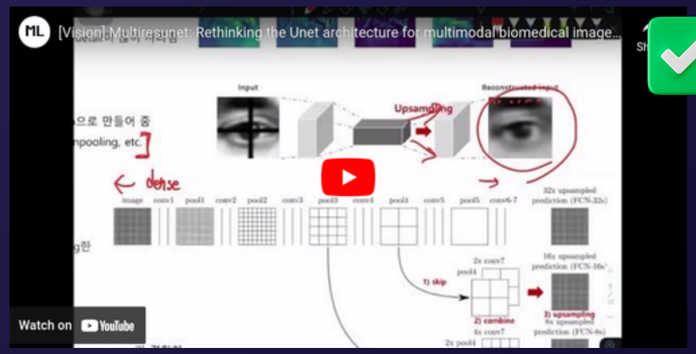


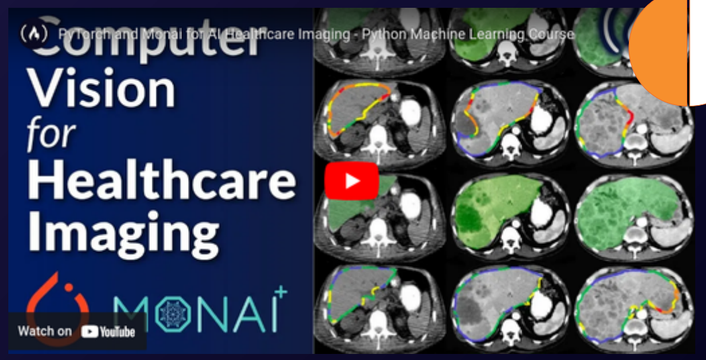




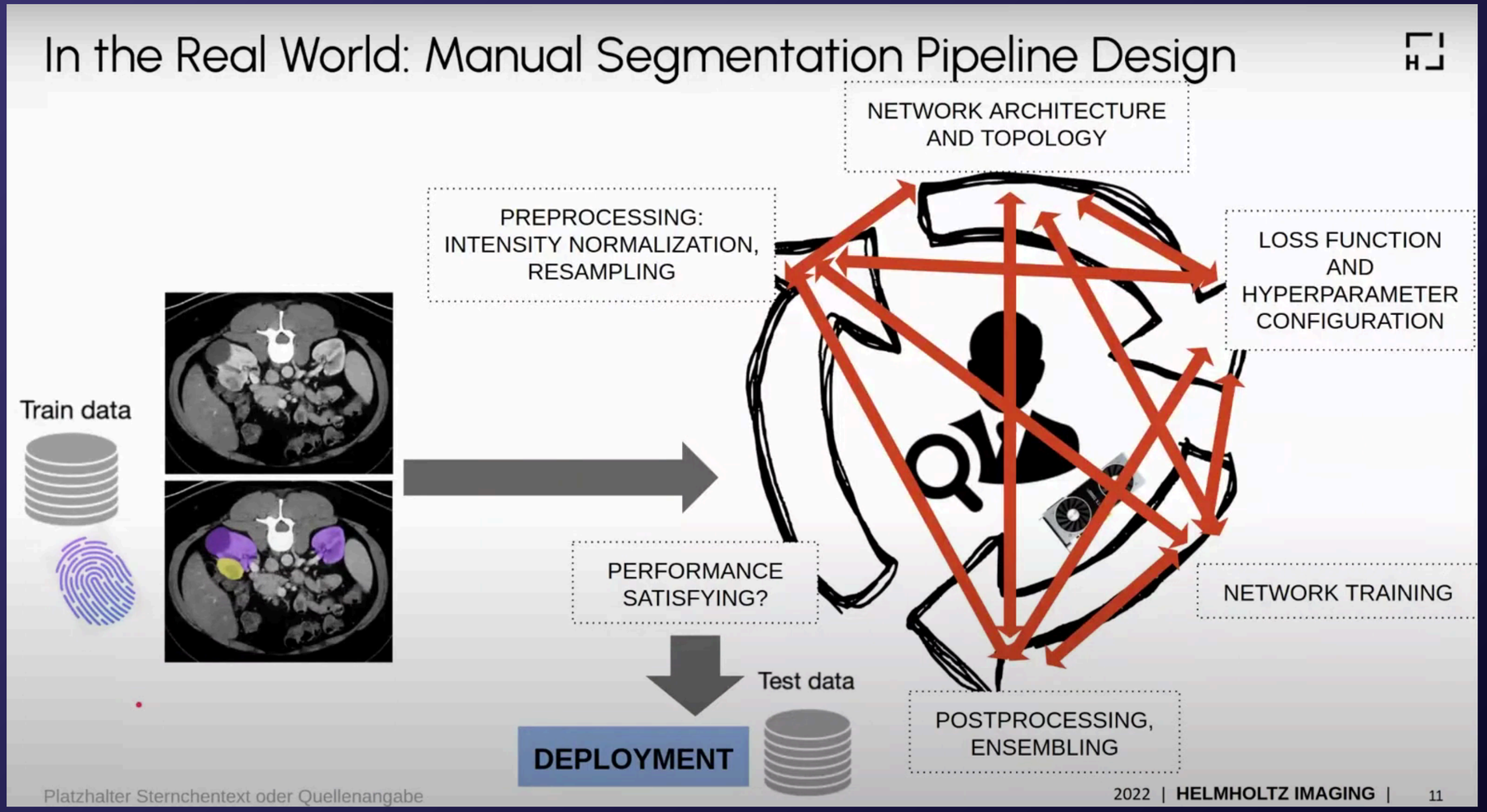








In the Real World: Manual Segmentation Pipeline Design



TransUnet Architecture

3D TransUnet Architecture

- Preprocessing
- Encoder
  - Tokenization and Embedding
  - MLP
  - MHSA
- Decoder

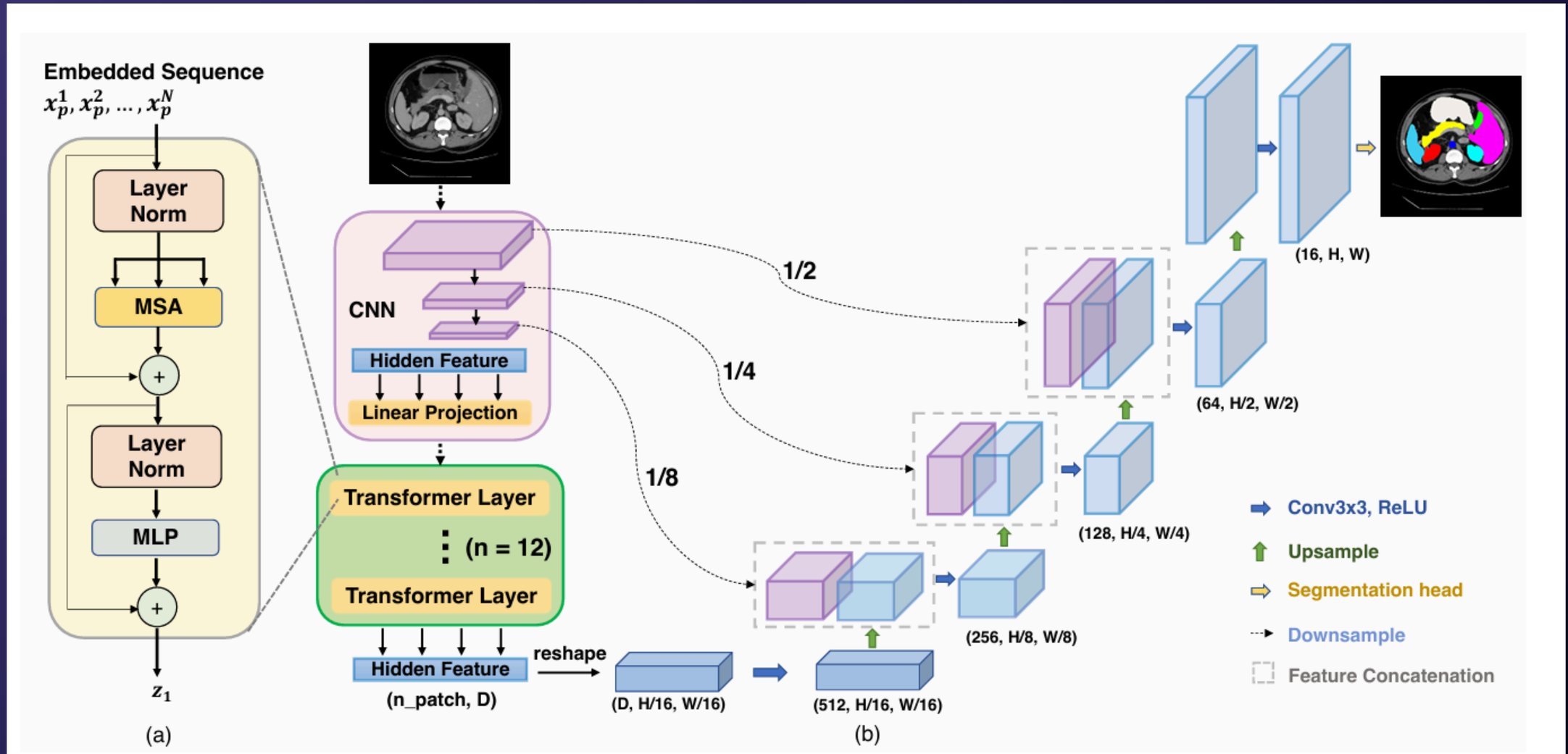
Loss Functions

Distance Transform

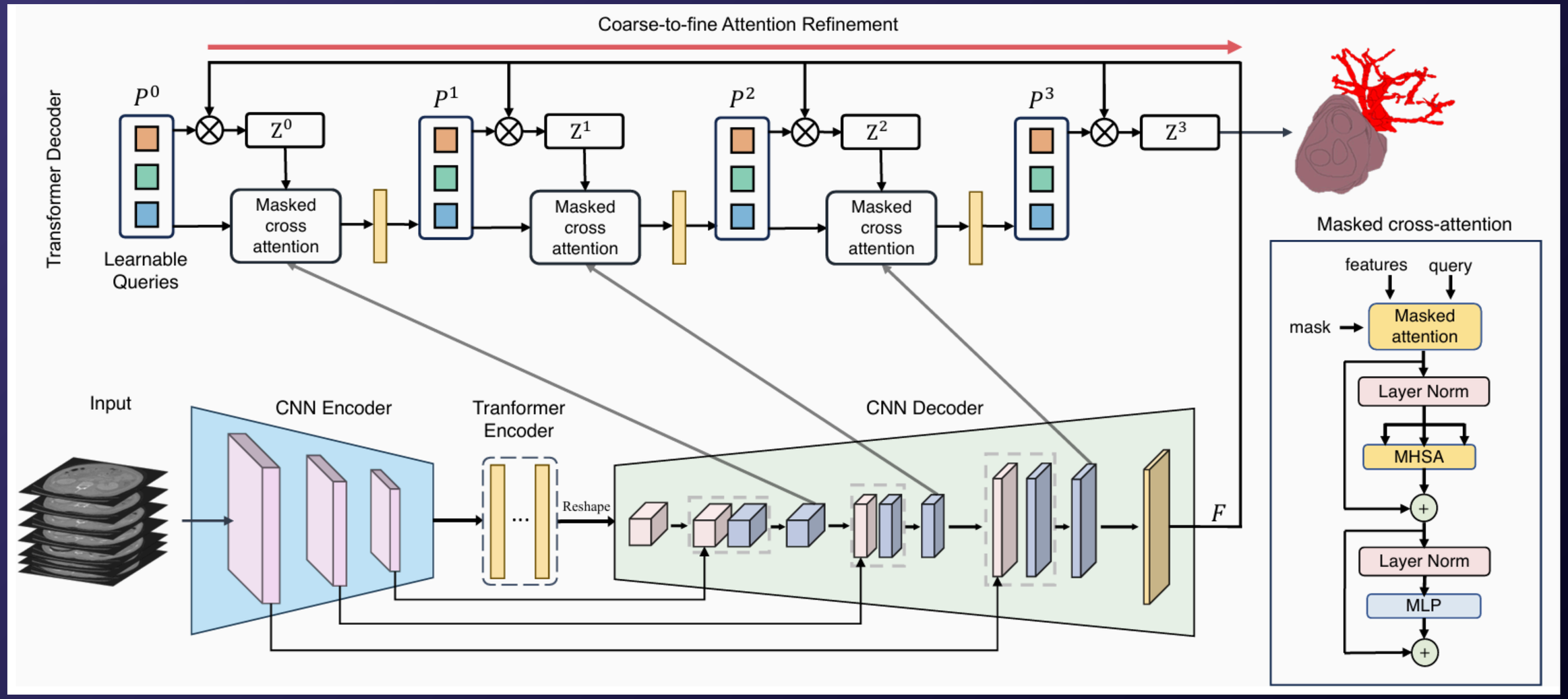
Code

- Data Format
- Preprocessing

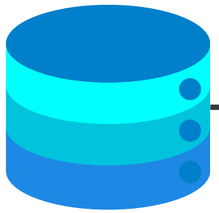
TransUnet Architecture



3D TransUnet Architecture

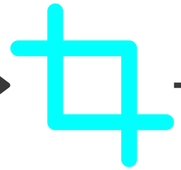


Preprocessing



Data Loading
.nii.gz

→Nonzero Mask Creation→Bounding Box Calculation→



Cropping

→Normalization and Resampling

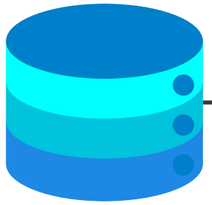
Data Augmentation

BrightnessMultiplicativeTransform
ContrastAugmentationTransform
GammaTransform
GaussianNoiseTransform
GaussianBlurTransform
SpatialTransform (like rotations, scaling, etc.)
SimulateLowResolutionTransform

Data Loading
.npz

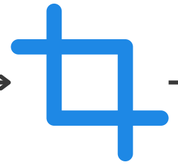
Identifiers Extraction

→On-the-fly Transformation to .npy→Patch Extraction



Data Loading
.nii.gz

→Nonzero Mask Creation→Bounding Box Calculation→



Cropping

→Normalization and Resampling

Data Augmentation

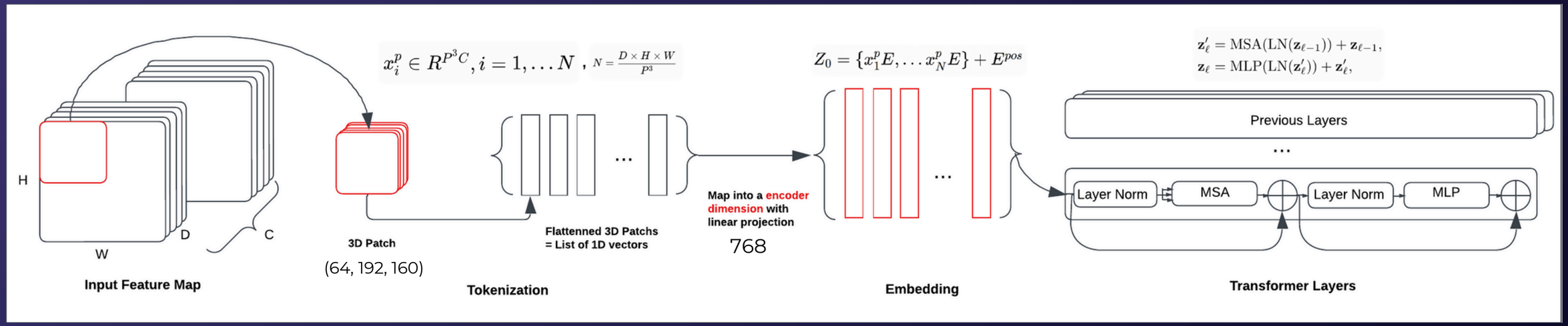
BrightnessMultiplicativeTransform
ContrastAugmentationTransform
GammaTransform
GaussianNoiseTransform
GaussianBlurTransform
SpatialTransform (like rotations, scaling, etc.)
SimulateLowResolutionTransform

Data Loading
(.npz)

Identifiers Extraction

→On-the-fly Transformation to .npy→Patch Extraction

Encoder



[batch_size, seq_length, 768]

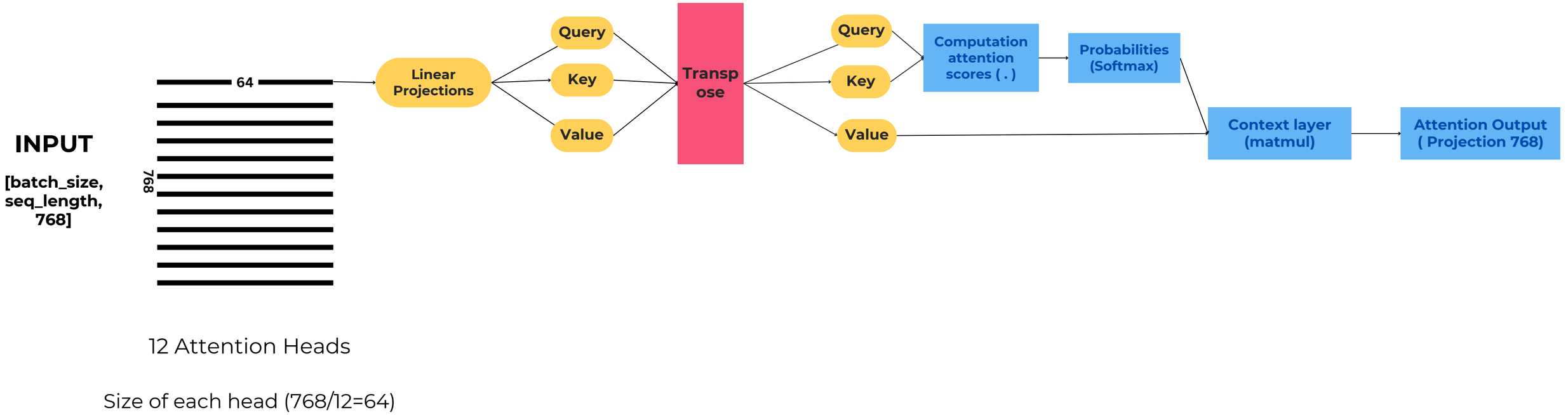
N° images , N° patches

Tokenization and Embedding

vit_modeling.py

MHSA

vit_modeling.py

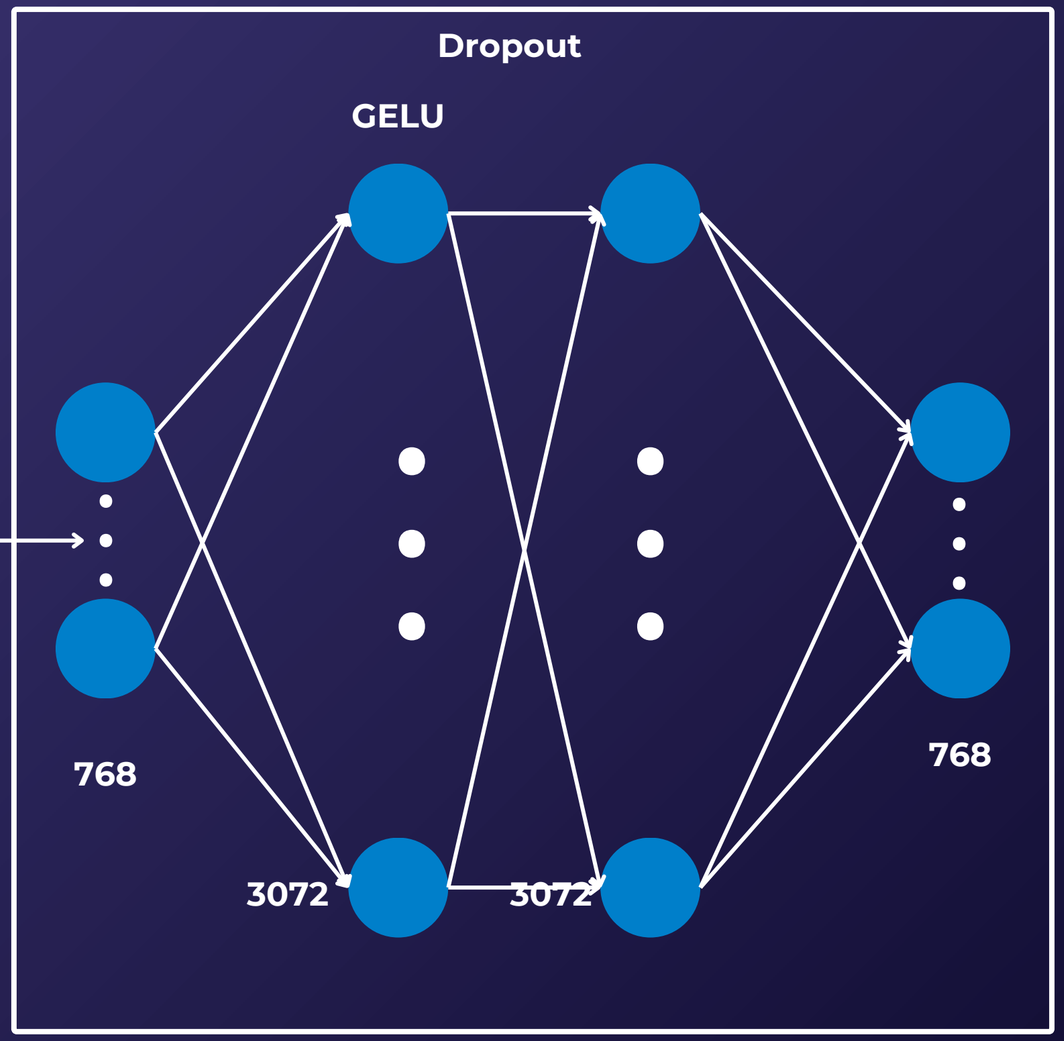


12 Attention Heads

Size of each head (768/12=64)

MLP

vit_modeling.py



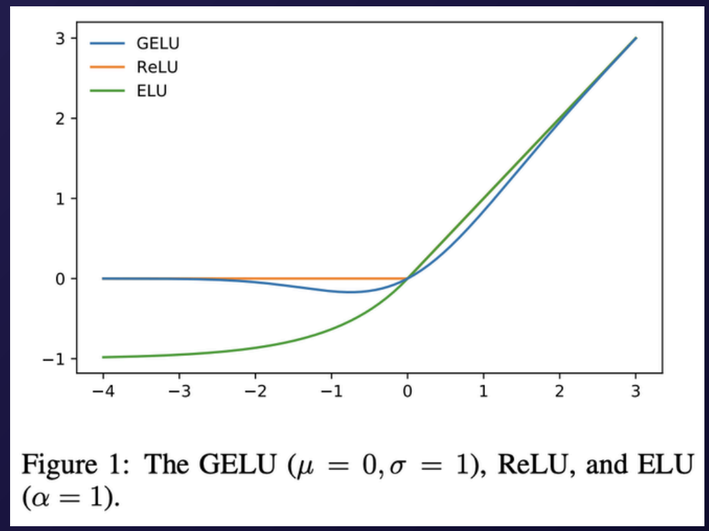
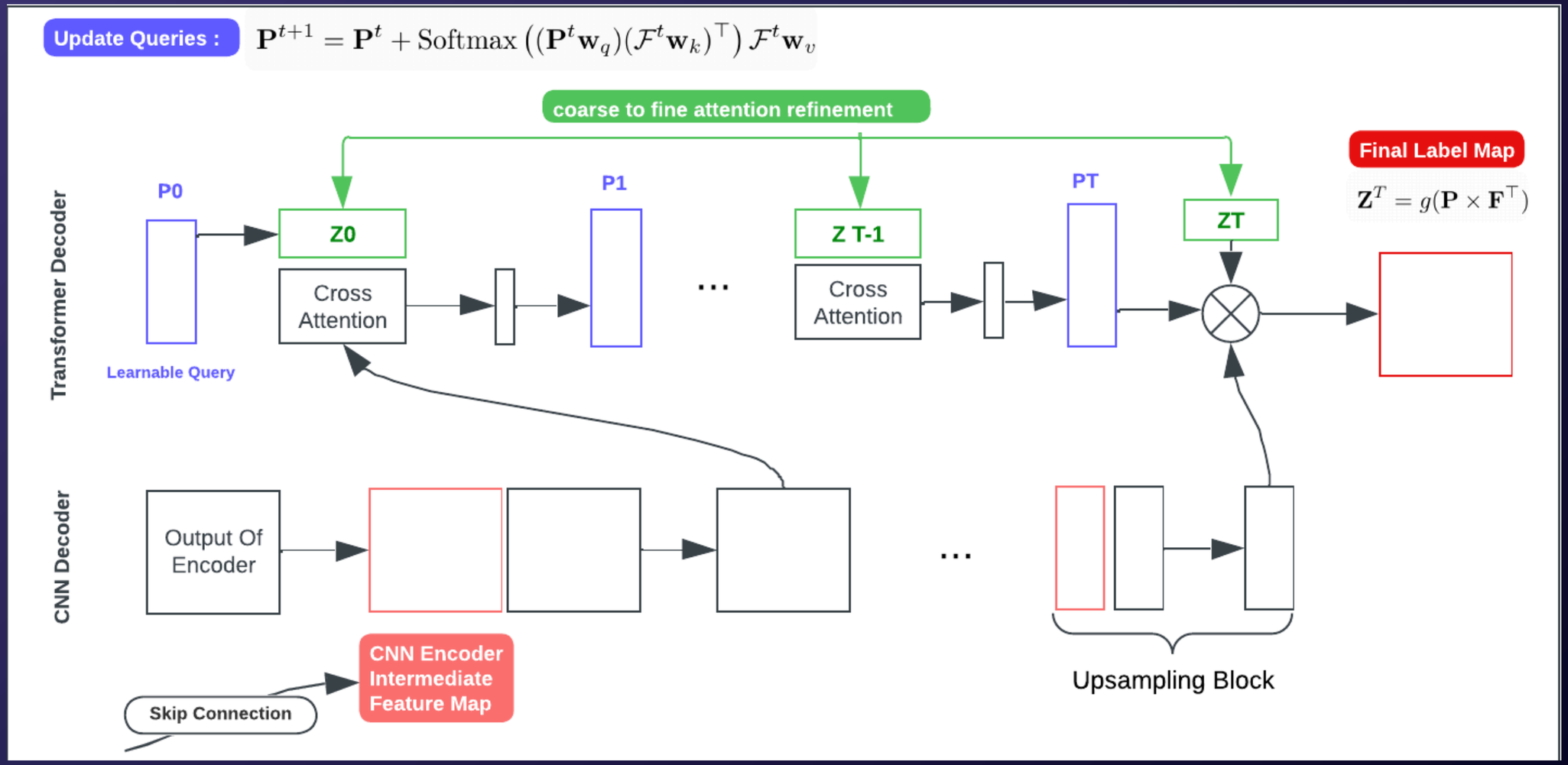


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

Decoder


Update Queries :
$\mathbf{P}^{t+1} = \mathbf{P}^t + \text{Softmax}\left((\mathbf{P}^t\mathbf{w}_q)(\mathcal{F}^t\mathbf{w}_k)^\top\right)\mathcal{F}^t\mathbf{w}_v$
coarse to fine attention refinement
Final Label Map
$\mathbf{Z}^T = g(\mathbf{P} \times \mathbf{F}^\top)$
Transformer Decoder
P0
P1
PT
Z0
Z T-1
ZT
Cross Attention
Cross Attention
Learnable Query
CNN Decoder
Output Of Encoder
CNN Encoder Intermediate Feature Map
Skip Connection
Upsampling Block

$$\mathcal{L} = \lambda_0(\mathcal{L}_{ce} + \mathcal{L}_{dice)} + \lambda_1\mathcal{L}_{cl}$$

Lcls represents the classification loss computed using the cross-entropy for each candidate region

Method	Lesion-wise Dice (↑)				HD95 (↓)			
	ET	TC	WT	Avg.	ET	TC	WT	Avg.
Encoder-only 3D-TransUNet [6]	54.79%	58.96%	56.05%	56.60%	108.9	107.6	109.5	108.7
Decoder-only 3D-TransUNet [6]	56.80%	61.12%	60.09%	59.34%	99.4	95.9	93.97	96.4

Table 1. Ablation Performance of 3D-TransUNet on the training set of BraTS-METS 2023 [17] under 5-fold cross-validation.

Dataset Split	Lesion-wise Dice (↑)				HD95 (↓)			
	ET	TC	WT	Avg.	ET	TC	WT	Avg.
Validation	59.2%	63.4%	56.5%	59.6%	94.8	94.8	110.9	100.1
Test	57.4%	62.0%	59.9%	59.8%	103.0	99.8	99.6	100.8

Table 2. Performance of Encoder-only 3D-TransUNet on the validation and test set of BraTS-METS 2023 [17].

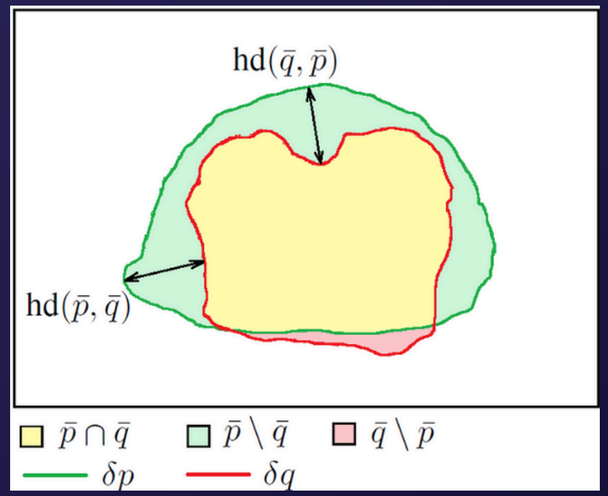
2019

$$DT_X[i,j] = \min_{k,l;X[k,l]=1} d([i,j],[k,l])$$

$$hd_{DT}(\delta p, \delta q) = \max_{\Omega}\left((\bar{p} \triangle \bar{q}) \circ d_q\right)$$

$$HD_{DT}(\delta q, \delta p) = \max\left(hd_{DT}(\delta q, \delta p), hd_{DT}(\delta p, \delta q)\right)$$

$$Loss_{DT}(q,p) = \frac{1}{|\Omega|}\sum_{\Omega}\left((p-q)^2 \circ (d_p^{\alpha} + d_q^{\alpha})\right)$$



emphasizes errors at the boundary of the segmentations

Binary

$$Loss_{DT\text{-}OS}(q,p) = \frac{1}{|\Omega|}\sum_{\Omega}\left((p-q)^2 \circ d_p^{\alpha}\right)$$

2021

2 labels & β =2

$$Loss_{mean}(q,p,L) = \frac{1}{|L|}\sum_{l\subset L}\frac{1}{|\Omega|}\sum_{\Omega}\left(q_l^{\beta}\cdot u_{l,p}^{\alpha}\right)$$

Exact Euclidean distance transform

$$Loss_{max}(q,p,L) = \frac{1}{|L|}\sum_{l\subset L} max_{\Omega}\left(q_l^{\beta}\cdot u_{l,p}^{\alpha}\right)$$

ul,p- the unidirectional distance from the border of the structure with label l outside the structure, and zero inside the structure.

mimic the sparse nature of the real Hausdorff distance more closely

$$Loss = W_{CE}\cdot Loss_{CE} + W_{mean}\cdot Loss_{mean} + W_{max}\cdot Loss_{max} + W_{VAE}\cdot Loss_{VAE} + W_{KL}\cdot Loss_{KL}$$

LossVAE- L2 norm of the variational autoencoder reconstruction error
LossKL- Kulback-Leibler norm of the difference of VAE parameters from the normal distributions with zero mean and unit standard deviation

Distance Transform

Ground Truth Segmentation Map

```
0 0 0 0
0 1 1 0
0 0 0 0
0 0 0 0
```

Euclidiean Distance Transform

```
1.414 1.000 1.000 1.414
1.000 0.000 0.000 1.000
1.000 1.000 1.000 1.414
2.236 2.000 2.000 2.236
```

Predicted Segmentation Map

```
0 0 0 0
0 1 0 0
0 1 0 0
0 0 0 0
```

$$\text{Loss}_{\text{ER}}(q,p) = \frac{1}{|\,\Omega\,|}\sum_{k=1}^{K}\sum_{\Omega}\left((p-q)^2 \ominus_k B\right)k^{\alpha}$$

successive erosions to the squared error between predicted and actual segmentations . **The largest "thickness" of the difference region that survives the erosion process approximates the HD**

$$\begin{aligned}\text{Loss}_{\text{CV}}(q,p) = \frac{1}{|\,\Omega\,|}\sum_{r\in R} r^{\alpha}\sum_{\Omega}\Big[&f_s(B_r * \bar{p}^C)\circ f_{\bar{q}\setminus\bar{p}} + \\ &f_s(B_r * \bar{p})\circ f_{\bar{p}\setminus\bar{q}} \; + \\ &f_s(B_r * \bar{q}^C)\circ f_{\bar{p}\setminus\bar{q}} + \\ &f_s(B_r * \bar{q})\circ f_{\bar{q}\setminus\bar{p}}\Big]\end{aligned}$$

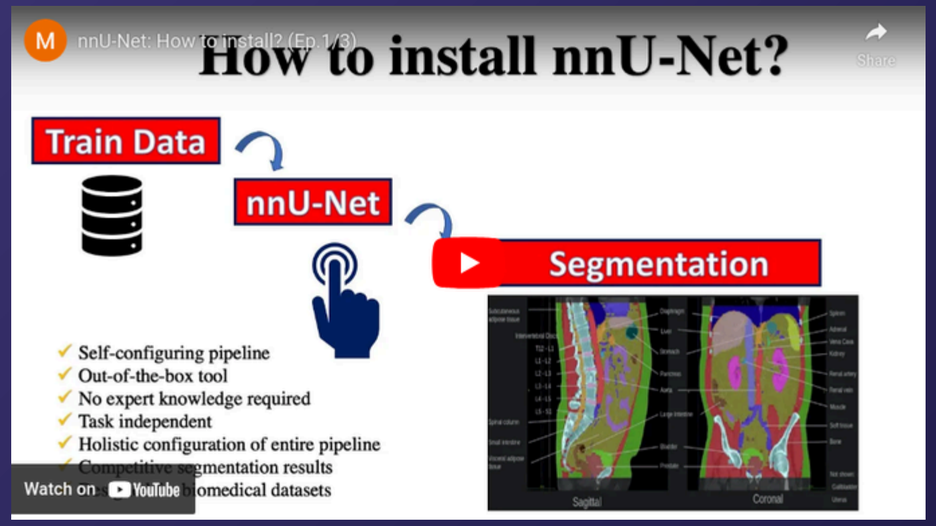
where $f_{\bar{q}\setminus\bar{p}}$ is a relaxed estimation of $\bar{q}\setminus\bar{p}$ defined as:

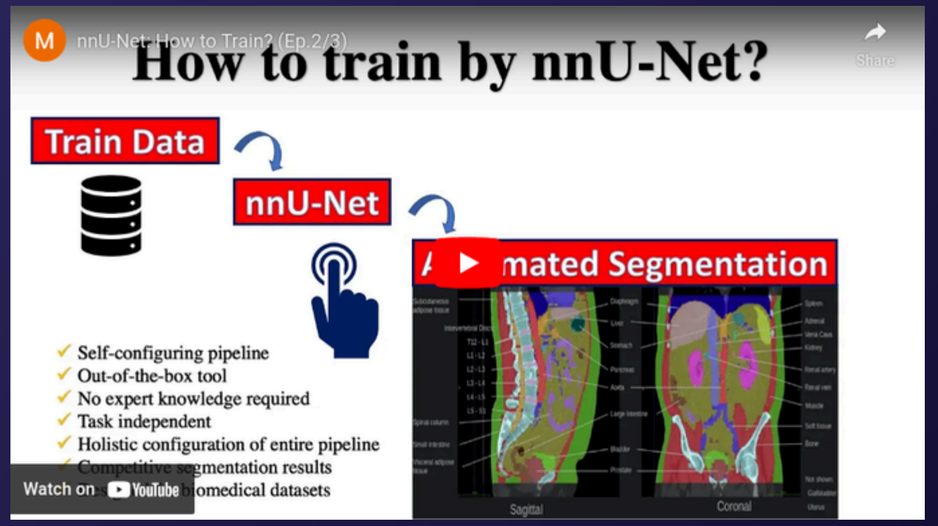
$$f_{\bar{q}\setminus\bar{p}} = (p-q)^2 q$$

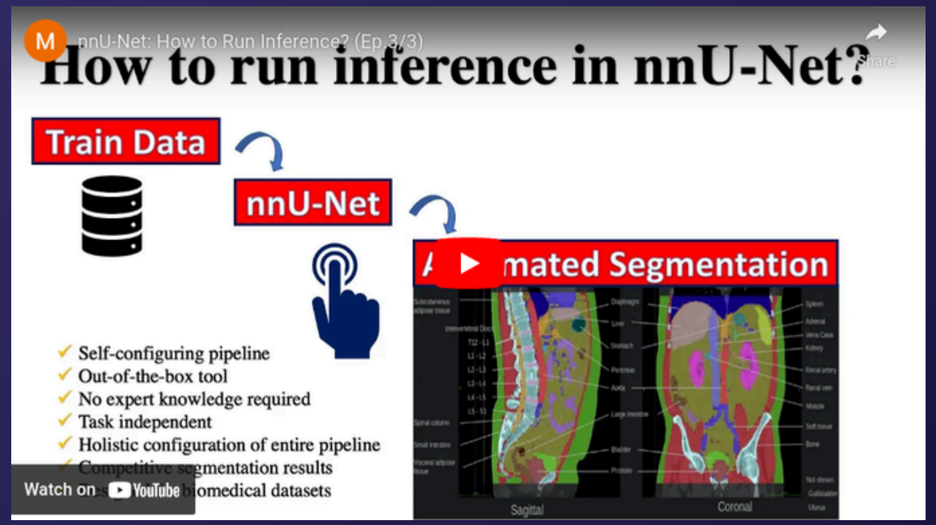
Uses **circular or spherical convolutional kernels** to measure how far the convolved image extends beyond the actual segmentation boundary.

The size of the kernel that still detects differences between the predicted and actual segmentations indicates the error.

It adjusts for **larger** errors more harshly, applying a scaling factor that increases with the kernel size.


nnU-Net: How to install? (Ep.1/3)
How to install nnU-Net?
Share
Train Data
nnU-Net
Segmentation
Self-configuring pipeline
Out-of-the-box tool
No expert knowledge required
Task independent
Holistic configuration of entire pipeline
Competitive segmentation results
Watch on YouTube

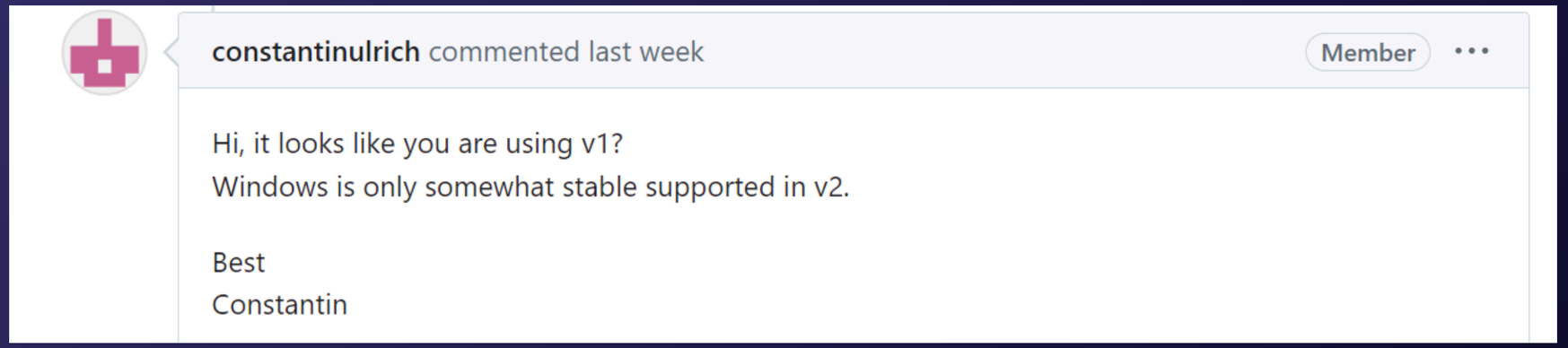

nnU-Net: How to Train? (Ep.2/3)
How to train by nnU-Net?
Share
Train Data
nnU-Net
mated Segmentation
Self-configuring pipeline
Out-of-the-box tool
No expert knowledge required
Task independent
Holistic configuration of entire pipeline
Competitive segmentation results
Watch on YouTube


nnU-Net: How to Run Inference? (Ep.3/3)
How to run inference in nnU-Net?
Share
Train Data
nnU-Net
mated Segmentation
Self-configuring pipeline
Out-of-the-box tool
No expert knowledge required
Task independent
Holistic configuration of entire pipeline
Competitive segmentation results
Watch on YouTube


nnUNet-V2 data pre-processing and training
Share
Watch on YouTube
Queen Mary
University of London

Reminder

Last Time, I attempted to do the pre-processing with nnUNetv1 but I've got an error that's apparently caused by the unstability of nnUNetv1 on windows



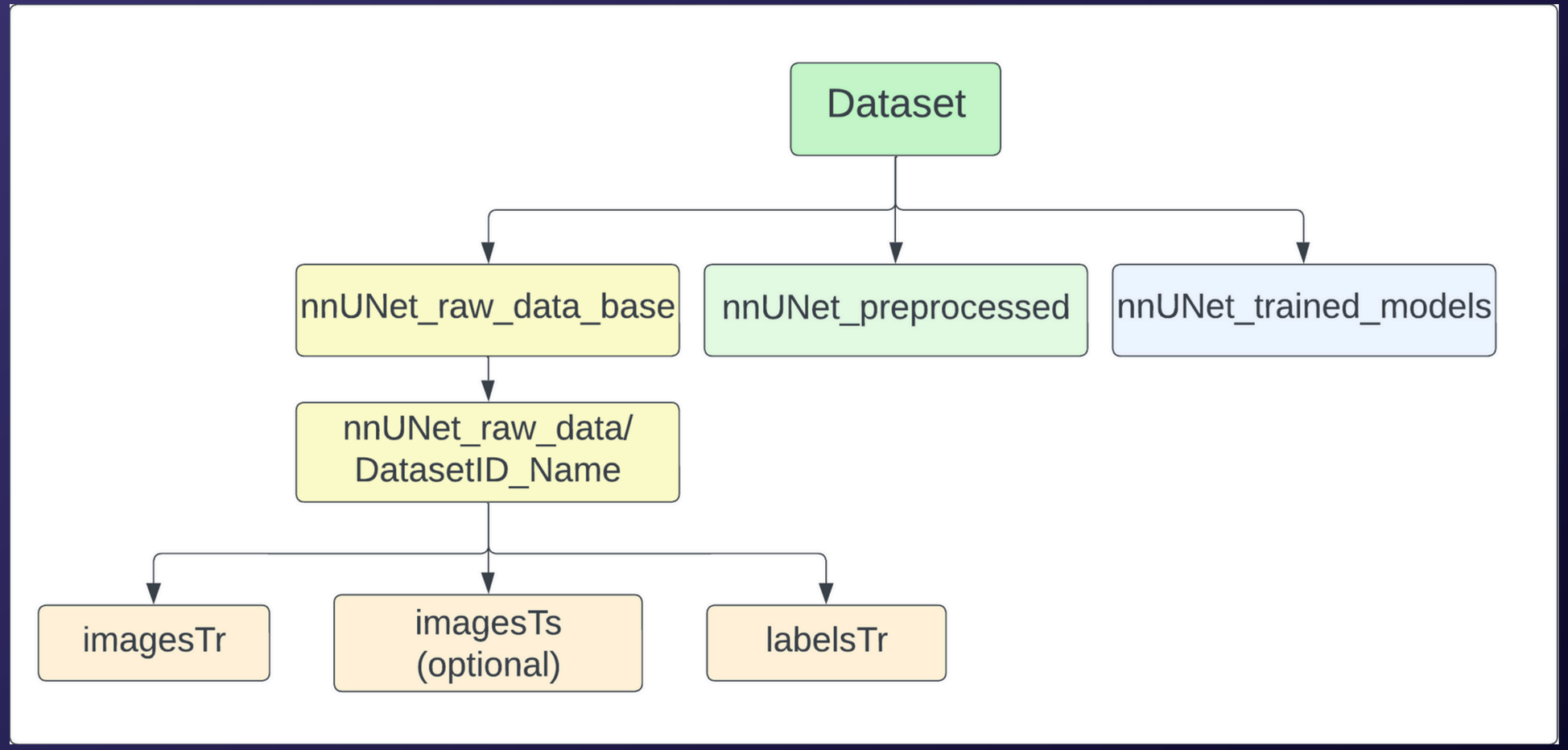
=> I've done this step with nnUNet v2

Issues:

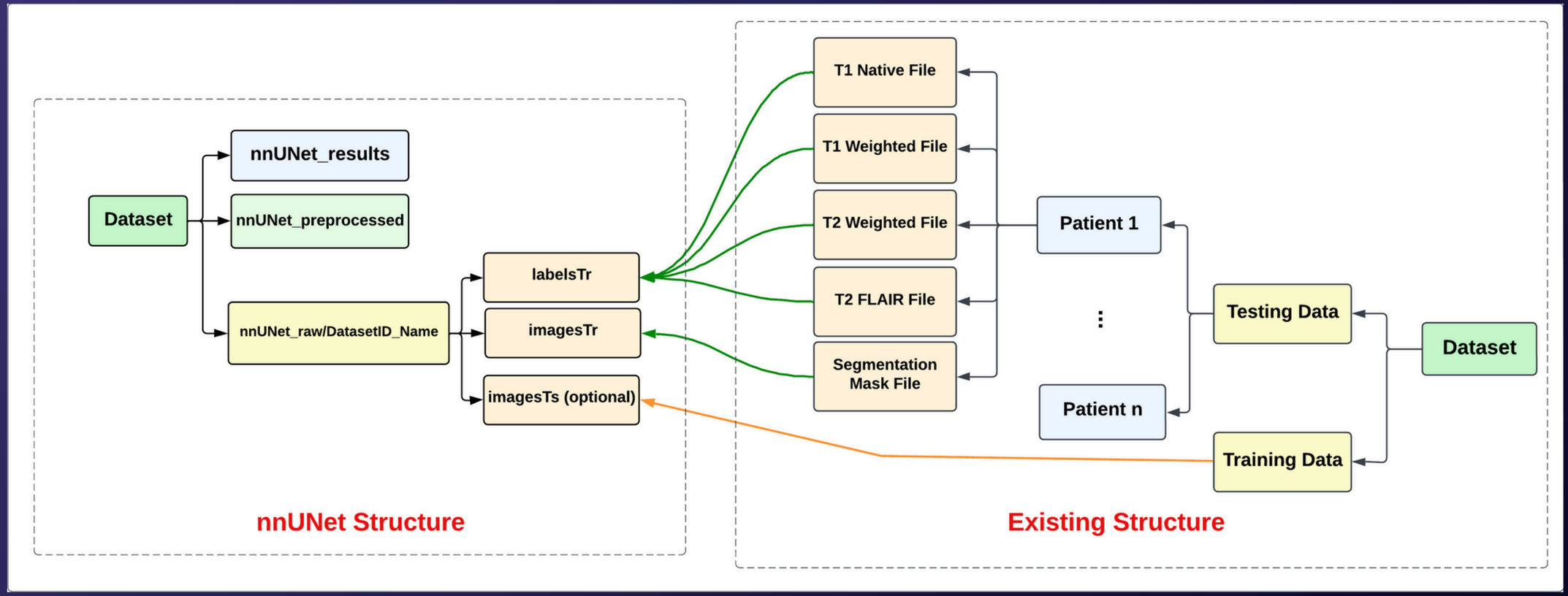
3D TransUNet uses nnUNetv1 specifically

=> Preprocessing will be done on kaggle since my ubuntu distribution can't handle this much data

Data Format


Dataset
nnUNet_raw_data_base
nnUNet_preprocessed
nnUNet_trained_models
nnUNet_raw_data/
DatasetID_Name
imagesTr
imagesTs
(optional)
labelsTr

Data Format

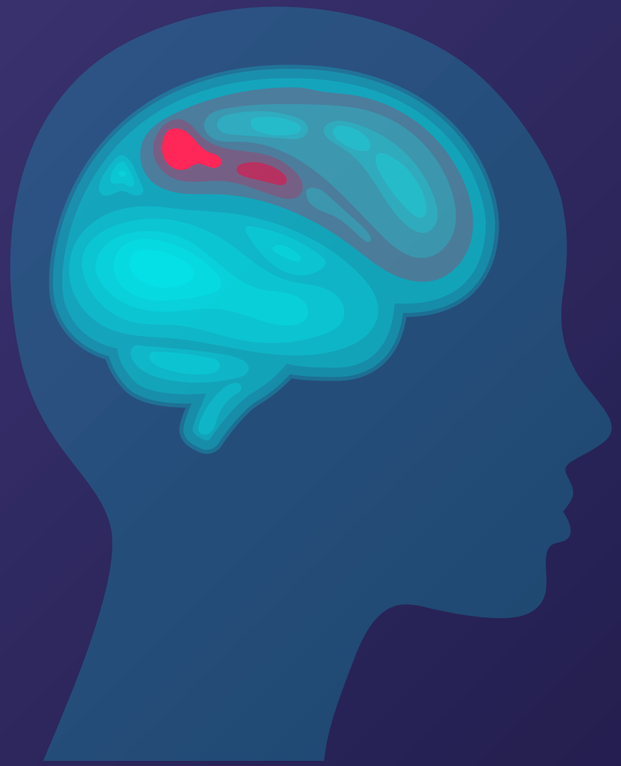

Dataset
nnUNet_results
nnUNet_preprocessed
nnUNet_raw/DatasetID_Name
labelsTr
imagesTr
imagesTs (optional)
nnUNet Structure
T1 Native File
T1 Weighted File
T2 Weighted File
T2 FLAIR File
Segmentation Mask File
Patient 1
Patient n
Testing Data
Training Data
Dataset
Existing Structure

Data Format

```
First/  165
 ├──Folder/
 │    ├── BraTS-MET-IDxx-000/
 │    │    ├── BraTS-MET-00810-t1n.nii.gz  # T1c
 │    │    ├── BraTS-MET-00810-t1c.nii.gz  # T1n
 │    │    ├── BraTS-MET-00810-t2w.nii.gz  # T2f
 │    │    ├── BraTS-MET-00810-t2f.nii.gz  # T2w
 │    │    ├── BraTS-MET-00810-seg.nii.gz
 │    ├── BraTS-MET-IDxx-000/
 │    │    ├── BraTS-MET-XXXX_0000.nii.gz
 │    │    ├── ...
 └──── ...
Additional/  73
...
Validation/ 31
...
```

```
nnUNet_raw/
 ├── Dataset180_BrainMets2023/
 │    ├── dataset.json
 │    ├── imagesTr/
 │    │    ├── BraTS-MET-00810_0000.nii.gz  # T1c
 │    │    ├── BraTS-MET-00810_0001.nii.gz  # T1n
 │    │    ├── BraTS-MET-00810_0002.nii.gz  # T2f
 │    │    ├── BraTS-MET-00810_0003.nii.gz  # T2w
 │    ├── labelsTr/
 │    │    ├── BraTS-MET-00810.nii.gz      # Segmentation
 │    ├── imagesTs/
 │    │    ├── BraTS-MET-XXXX_0000.nii.gz
 │    │    ├── ...
 └──── ...
```

checkfile.py & checkfile2.py



BraTS-MET-00404-000_t2w.nii.gz	20/07/2024 14:08	WinRAR	5,705 KB
BraTS-MET-00404-000_t2f.nii.gz	20/07/2024 14:08	WinRAR	5,695 KB
BraTS-MET-00404-000_t1n.nii.gz	20/07/2024 14:08	WinRAR	5,626 KB
BraTS-MET-00404-000_t1c.nii.gz	20/07/2024 14:08	WinRAR	5,612 KB
BraTS-MET-00404-000_.nii.gz	20/07/2024 14:08	WinRAR	7 KB

```
PS C:\Users\makni\Downloads\brain metastases\3DTransUNet\3D-TRANSUNet\Brats2023>
ckfiles2.py
Patient ID 00404 has an issue: 5
PS C:\Users\makni\Downloads\brain metastases\3DTransUNet\3D-TRANSUNet\Brats2023>
ckfiles2.py
All patient files in imagesTr are correctly organized.
```

dataset.json Generation

To simplify the steps and work on the pipeline, I uploaded a sample of the data : 5patient samples for training (since nnUNet works with 5 fold) and 2 for test


```
{
    "description": "BRATS Met Brain Tumour Segmentation Dataset",
    "labels": {
        "0": "background",
        "1": "necrotic and non-enhancing tumor core",
        "2": "edema",
        "3": "enhancing tumor"
    },
    "licence": "hehe",
    "modality": {
        "0": "T1",
        "1": "T1ce",
        "2": "T2",
        "3": "Flair"
    },
    "name": "Task180_BraTSMet",
    "numTest": 1,
    "numTraining": 5,
    "reference": "http:..",
    "release": "1.0",
    "tensorImageSize": "4D",
    "test": [
        "./imagesTs/BraTS-MET-00002-000.nii.gz"
    ],
    "training": [
        {
            "image": "./imagesTr/BraTS-MET-00002-000.nii.gz",
            "label": "./labelsTr/BraTS-MET-00002-000.nii.gz"
        },
        {
            "image": "./imagesTr/BraTS-MET-00005-000.nii.gz",
            "label": "./labelsTr/BraTS-MET-00005-000.nii.gz"
        },
```

Preprocessing

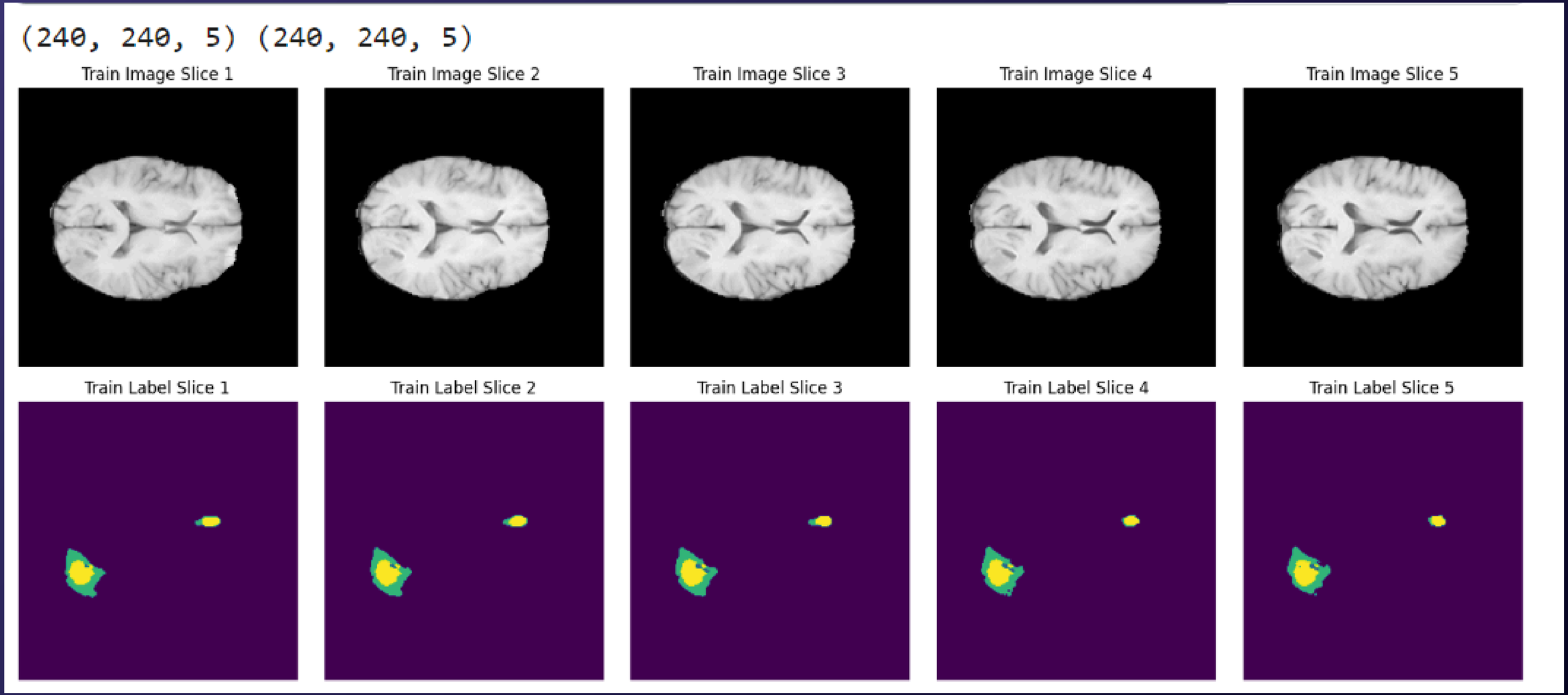
```
import os
os.chdir(main_dir)
!python .../nnUNet_plan_and_preprocess.py -t 180 --verify_dataset_integrity
os.chdir(base_dir)
```

```
normalization...
no resampling necessary
no resampling necessary
before: {'spacing': array([1., 1., 1.]), 'spacing_transposed': array([1., 1., 1.]), 'data.shape (data is transposed)': (4, 132, 167, 124)}
after:  {'spacing': array([1., 1., 1.]), 'data.shape (data is resampled)': (4, 132, 167, 124)}

normalization...
normalization done
normalization done
2 1533
3 1578
saving:  /kaggle/working/nnUNet/nnunet/preprocessed/Task180_BraTSMet/nnUNetData_plans_v2.1_2D_stage0/BraTS-MET-00005-000.npz
normalization done
normalization done
normalization done
1 689
2 267
3 487
saving:  /kaggle/working/nnUNet/nnunet/preprocessed/Task180_BraTSMet/nnUNetData_plans_v2.1_2D_stage0/BraTS-MET-00089-000.npz
2 10000
3 5073
saving:  /kaggle/working/nnUNet/nnunet/preprocessed/Task180_BraTSMet/nnUNetData_plans_v2.1_2D_stage0/BraTS-MET-00530-000.npz
1 6744
1 465
2 10000
2 10000
3 10000
saving:  /kaggle/working/nnUNet/nnunet/preprocessed/Task180_BraTSMet/nnUNetData_plans_v2.1_2D_stage0/BraTS-MET-00002-000.npz
3 8767
saving:  /kaggle/working/nnUNet/nnunet/preprocessed/Task180_BraTSMet/nnUNetData_plans_v2.1_2D_stage0/BraTS-MET-00379-000.npz
```

Dropping 3d_lowres config because the image size difference to 3d_fullres is too small

Data Visualisation



Train nnUNet 10 epochs

```
!nUNos.chdir(main_dir)
!CUDA_VISIBLE_DEVICES=0 nnUNet_train 3d_fullres nnUNetTrainerV2 180 0
os.chdir(base_dir)
```

Debugging Steps:

APEX Library: Apex is a set of tools developed by NVIDIA that enable easier and faster mixed precision and distributed training in PyTorch. Mixed precision training involves using both 16-bit and 32-bit floating-point types during training to make it faster and more memory efficient without sacrificing the model's accuracy or stability.

Configuration Parameters nnUNet

Parameter	Value
Trainer Class	nnUNetTrainerV2
Number of Classes	3
Modalities	T1, T1ce, T2, Flair
Normalization Schemes	nonCT for all modalities
Batch Size (Stage 0)	2
Num Pool Per Axis	[5, 5, 4]
Patch Size	[128, 160, 112]
Median Patient Size	[138, 174, 132]
Current Spacing	[1.0, 1.0, 1.0]
Original Spacing	[1.0, 1.0, 1.0]
Data Folder	/kaggle/working/nnUNet/nnunet/preprocessed/Task180_BraTSMet/nnUNetData_plans_v2.1

nnUNet Results on 10 epochs for only 5 samples

Epoch	Learning Rate	Train Loss	Validation Loss	Avg. Dice Class 1	Avg. Dice Class 2	Avg. Dice Class 3	Epoch Duration (s)
0	0.009991	-0.1617	-0.4018	0.0	0.7285	0.805	856.33
1	0.009982	-0.5570	-0.3314	0.0	0.7147	0.7199	947.24
2	0.009973	-0.7192	-0.3213	0.0013	0.8207	0.6316	891.90
3	0.009964	-0.7546	-0.1713	0.0	0.6402	0.6282	838.72
4	0.009955	-0.7631	-0.3217	0.0017	0.7311	0.7247	875.59
5	0.009946	-0.7978	-0.3323	0.0	0.7087	0.7654	931.70
6	0.009937	-0.8010	-0.2978	0.0003	0.7152	0.6658	869.26
7	0.009928	-0.8049	-0.3802	0.0004	0.7412	0.7706	934.93
8	0.009919	-0.8127	-0.3502	0.0006	0.7077	0.7605	873.78
9	0.00991	-0.8159	-0.0555	0.0002	0.4975	0.6333	1065.26
10	0.009901	-0.8227	0.0307	0.0	0.5623	0.3309	1029.19

- Typically nnUNet runs 1000 epochs
- Each epoch takes >15 minutes + ~15/16GiB CPU and ~8GiB GPU

Training with 3D TransUNet Presented Multiple Issues

```
Running distributed on rank 0 of 1
Network: 3d_fullres
Task: Task180_BraTSMet
Network Trainer: nnUNetTrainerV2_DDP
Plans Identifier: nnUNetPlansv2.1
HDFS Base: GeTU500Region_3DTransUNet_decoder_only
Plan Update:
['/kaggle/working/nnUNet/nnunet/nnUNet_raw_data_base/nnUNet_raw_data/training/network_training'] nnUNetTrainerV2_DDP nnunet.trainin
g.network_training
################################################
I am running the following nnUNet: 3d_fullres
My trainer class is:  <class 'nn_transunet.trainer.nnUNetTrainerV2_DDP.nnUNetTrainerV2_DDP'>
For that I will be using the following configuration:
I am using stage 0 from these plans
I am using sample dice + CE loss

I am using data from this folder:  /kaggle/working/nnUNe
################################################
    File "/opt/conda/lib/python3.10/site-package
      return launch_agent(self._config, self._en
    File "/opt/conda/lib/python3.10/site-package
      raise ChildFailedError(
  torch.distributed.elastic.multiprocessing.erro
  ============================================
  ./train.py FAILED
```

KeyError on Single GPU Setup and Clarification on Rank Environment Variable #31

Edit | New issue

Open · mariem-m11 opened this issue 4 minutes ago · 0 comments

mariem-m11 commented 4 minutes ago • edited

I am attempting to train using a single GPU setup. I modified the config file to utilize `nnUNetTrainerV2` instead of `nnUNetTrainerV2_DDP`.

And modified the train.sh like so:
nnunet_use_progress_bar=1 CUDA_VISIBLE_DEVICES=0 torchrun ./train.py --task="Task180_BraTSMet" --fold=${fold} --config=$CONFIG --network="3d_fullres" --resume='' --local-rank=0 --optim_name="adam" --valbest --val_final --npz

The script throws a `KeyError` when trying to access the `patch_size` from the `plan_data['plans']['plans_per_stage'][resolution_index]`.

Error Message:

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

3D TransUNet Modifications

Non Distributed Training

3D TransUNet is designed to work on distributed system (HDFS).
=> make it work in a single-process (non-distributed) environment.

Modifications

- The original command in 'train.sh' included the following:

```
nnunet_use_progress_bar=1 CUDA_VISIBLE_DEVICES=0 torchrun ./train.py --task="Task180_BraTSMet" --fold=${fold} --
config=$CONFIG --network="3d_fullres" --resume='' --optim_name="adam" --valbest --val_final --npz
```

- Error related to 'patch_size' . The code attempted to access a non-existent key in 'plans_per_stage', causing a failure. Added the following code:

```
print("Available keys in plans_per_stage:", plan_data['plans']['plans_per_stage'].keys())

 available_keys = plan_data['plans']['plans_per_stage'].keys()
 if 1 not in available_keys:
 print(f"Key 1 not found in plans_per_stage, using key {list(available_keys)[0]}")
 resolution_index = list(available_keys)[0]
 patch_size = plan_data['plans']['plans_per_stage'][resolution_index]['patch_size']
```

3D TransUNet Modifications

- Removed the block of code responsible for initializing distributed training:
- Also removed the arguments related to distributed training from the argument parser:

```
if 'WORLD_SIZE' in os.environ:
torch.distributed.init_process_group(backend='nccl', init_method='env://')
args.world_size = torch.distributed.get_world_size()
args.rank = torch.distributed.get_rank()
torch.distributed.barrier()
print(f"Running distributed on rank {args.rank} of {args.world_size}")
else:
args.world_size = 1
args.rank = 0
print("Running in non-distributed mode.")
```

```
parser.add_argument("--local-rank", type=int, default=0, help="Local rank. Necessary for using the
torch.distributed.launch utility.")
parser.add_argument('--world-size', default=1, type=int, help='number of nodes for distributed training')
parser.add_argument('--rank', default=0, type=int, help='node rank for distributed training')
```

- The network Trainer needed to be changed to 'nnUNetTrainerV2' and adapted for the task.

```
from nn_transunet.trainer.loss_functions import DC_and_CE_loss

# Removed these two lines
self.print_to_log_file("TRAINING KEYS:\n %s" % (str(self.dataset_tr.keys())), also_print_to_console=False)
self.print_to_log_file("VALIDATION KEYS:\n %s" % (str(self.dataset_val.keys())), also_print_to_console=False)
```

3D TransUNet Modifications

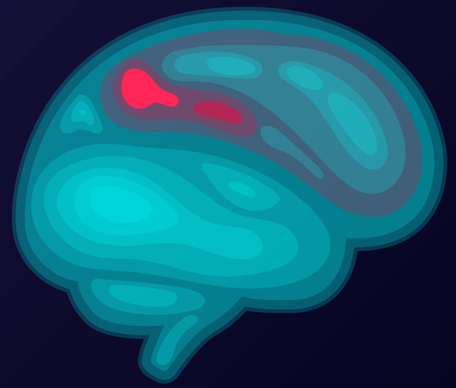
Loss Function Computation Error

During the training loop, an error occurred in the loss function computation:
Error: 'dict' object has no attribute 'shape'

Proposed Solution

Adaptation from `nnUNetTrainerV2_DDP`: The distributed version of the trainer (`nnUNetTrainerV2_DDP`) has logic to handle complex outputs and multi-stage deep supervision. The idea is to adapt similar logic to `nnUNetTrainerV2`:

- Modify the `run_iteration` and `compute_loss` methods to correctly unpack and handle the model's outputs, ensuring the loss function receives the appropriate tensors.
- Handle cases where the model outputs dictionaries by extracting the necessary values and passing them to the loss function in the expected format.



Key Differences and Adaptations for 3D TransUNet Trainer (DDP):

- **Initialization and Configuration:**

Additional parameters are initialized such as layer_decay, lr_scheduler_name, and several model-specific flags (e.g., disable_ds, is_spatial_aug_only).
The initialize_network method in 3D TransUNet includes the setup for a transformer-based model (Generic_TransUNet_max_ppbp) with specific settings for convolution operations and normalization layers.

- **Data Augmentation:**

A new method setup_DA_params_BraTSRegions specifically tailored for brain tumor segmentation (BraTS) is added

- **Distributed Training Adaptations:**

The 3D TransUNet version includes additional configurations to manage batch sizes across GPUs and specific conditions for different training strategies and optimizations.

- **Loss Computation and Backpropagation:**

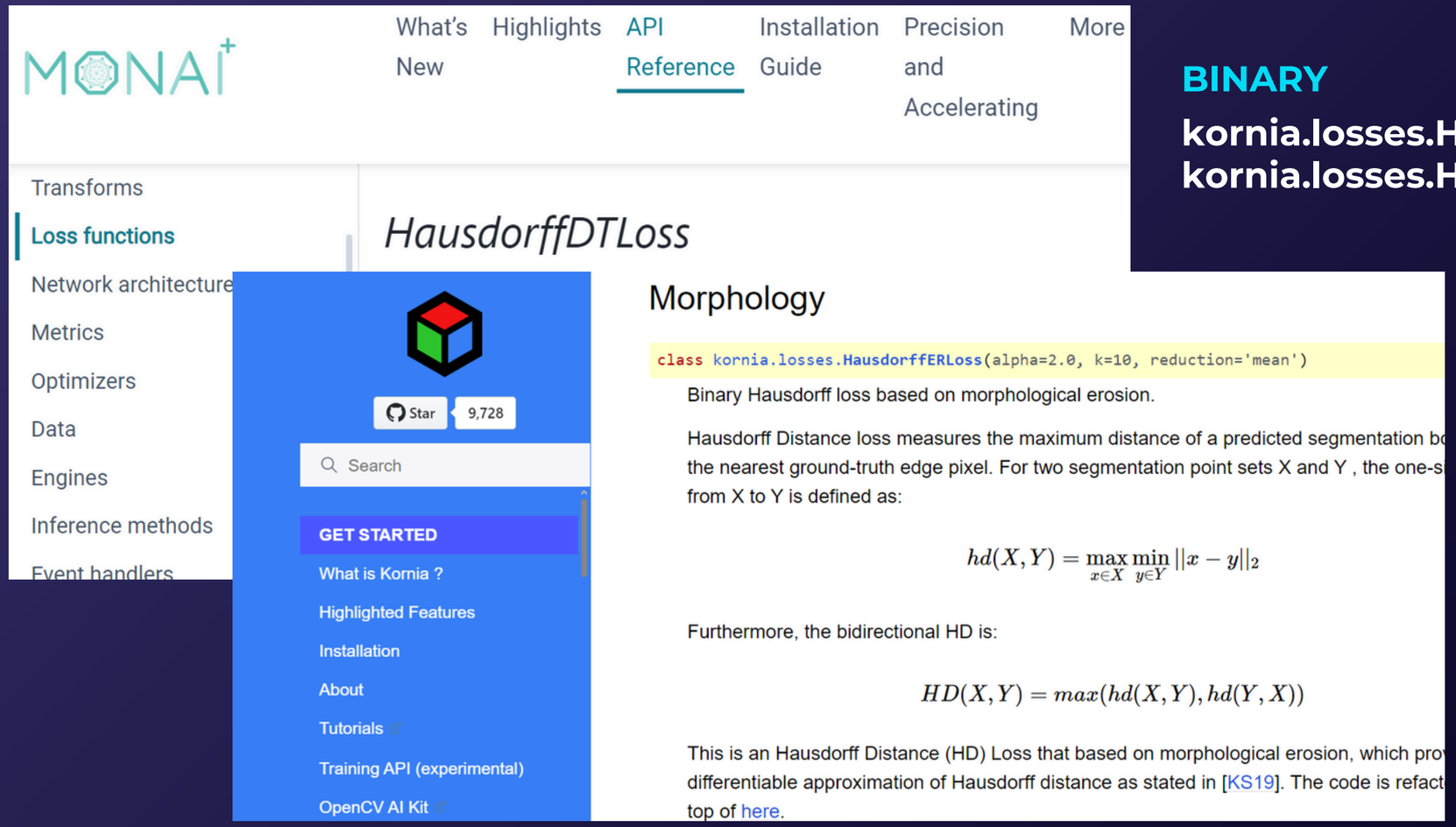
The loss computation in 3D TransUNet is extended to accommodate transformer outputs and potentially multiple outputs (deep supervision). The method compute_loss is significantly more complex in the 3D TransUNet version, including handling of various flags and configurations specific to the model architecture.
The handling of gradients and application of backpropagation includes checks for NaN values and additional debugging outputs.

- **Optimizer and Scheduler:**

Extended configurations for optimizers and learning rate schedulers are provided: warmup phases, custom learning rate schedules...

Hausdorff loss implementation based on the article

has been integrated into the following libraries:



BINARY

kornia.losses.HausdorffERLoss
kornia.losses.HausdorffERLoss3D

Extending to Multi-label Segmentation

We can compute the Hausdorff distance for each label separately and then aggregate these distances.

- **Target and Prediction Tensors:** Each label should be treated as a separate binary segmentation task within the overall calculation.

  → For each label, create a binary mask from the multi-label segmentation maps of both predictions and targets.

- **Compute Loss for Each Label: L**oop over each label, apply the binary loss function, and store the results.

- **Aggregate the Losses:**
  - **Sum:** Add up the losses across all labels to get a total loss for the batch.
  - **Mean:** Compute the mean of these losses to normalize the loss across labels.

Next Steps

Adaptation of trainer class to non distributed environment (.)

Adaptation of Hausdorff loss function to multilabel (CV/ ER)

Integrating the new loss function class

Training