

FeTS Segmentation & Label Fusion

Federated Tumor Segmentation

FeTS Segmentation & Label Fusion Wireframe

Local Disk (C:)

FeTS

TrainedModels

ACSA1

201810311130.yml

ACSA2

ACSA3

201809100922.yml

201810311015.yml

201812031137.yml

ACSA4

ACSA5

ACSA6

ACSA7

ACSA8

ACSA9

201810310924.yml

201810311015.yml

A

Input Images

Pre-processing

Segmentation

Training and Model Updates

Load Pre-Trained Models

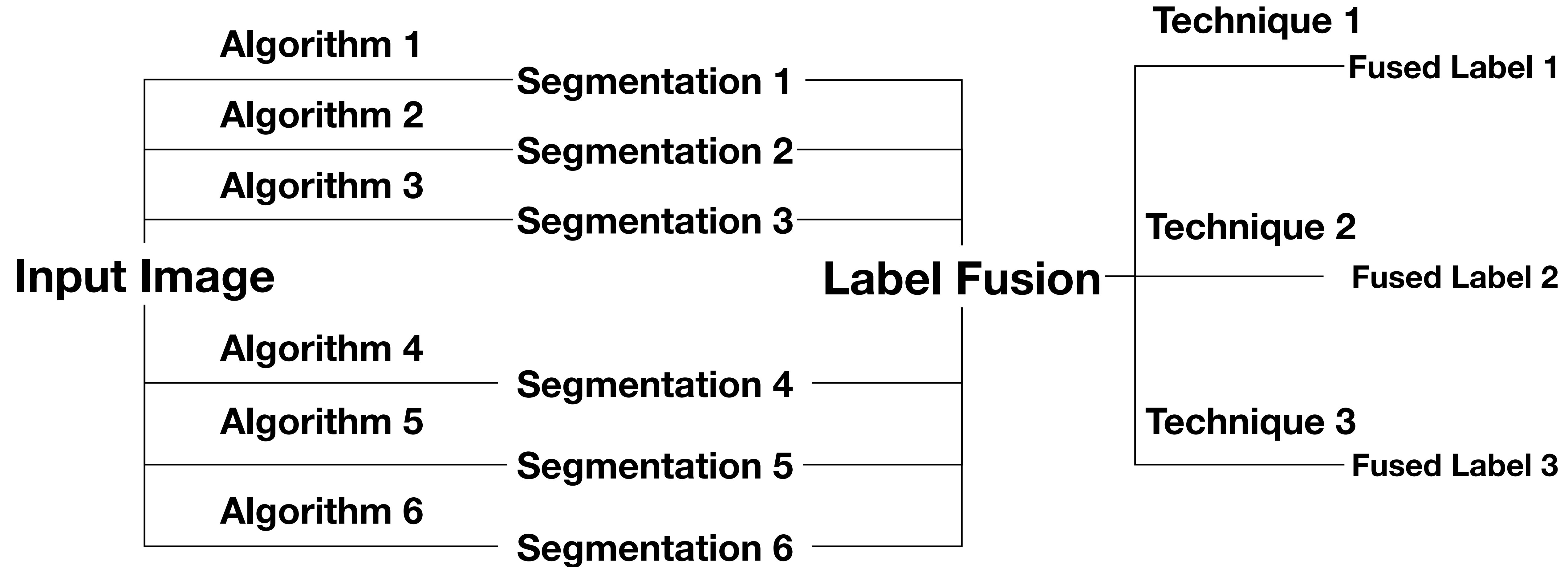
Fusion strategy drop-down menu

Fusion of ACSAs

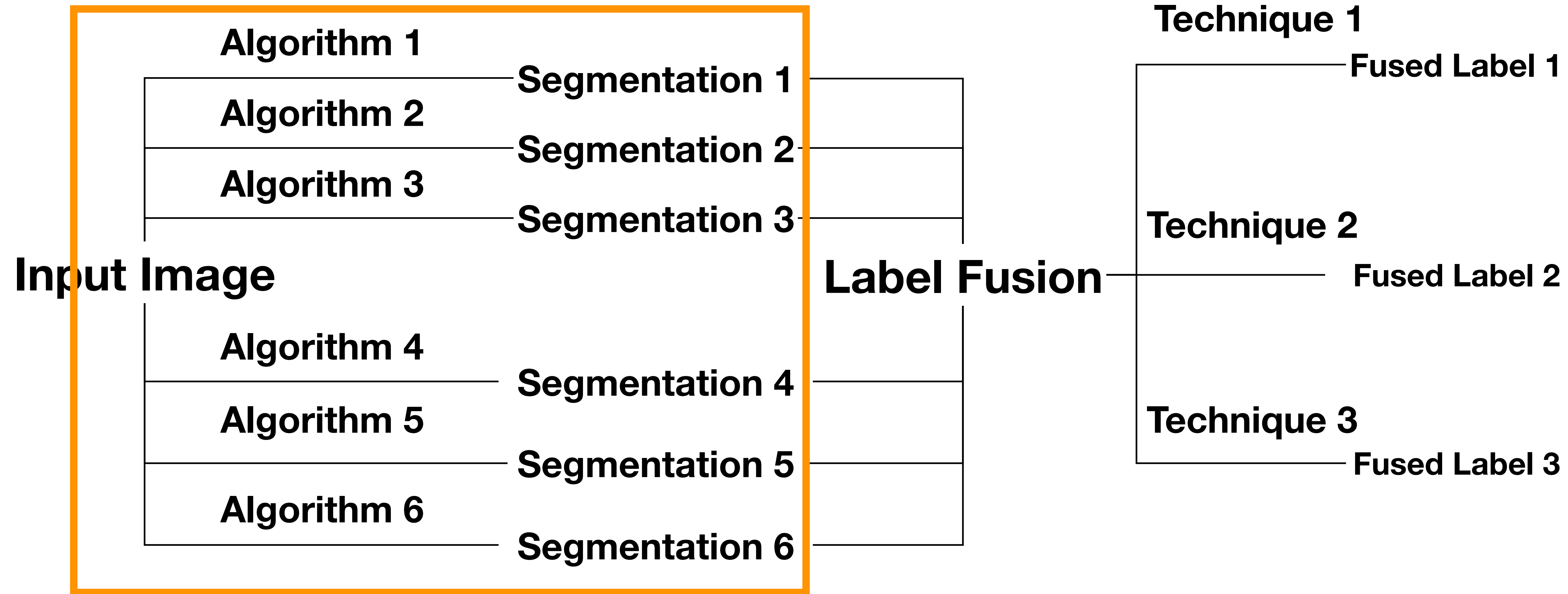
Algorithm	Model Path	Relevant Meta-Data
ACSA 1	C:/FeTS/TrainedModels/ACSA1/201810311130.yml	Training subjects: 1000, Arch Type: U-Net
ACSA 3	C:/FeTS/TrainedModels/ACSA3/201810301015.yml	Training subjects: 1000, Arch Type: ResNet
ACSA 9	C:/FeTS/TrainedModels/ACSA9/201810301015.yml	Training subjects: 2500, Arch Type: DeepMedic

B

Basic FeTS Concept



Basic FeTS Concept



Segmentation algorithms :

1. In-house

**2. BraTS Algorithmic
Repository**

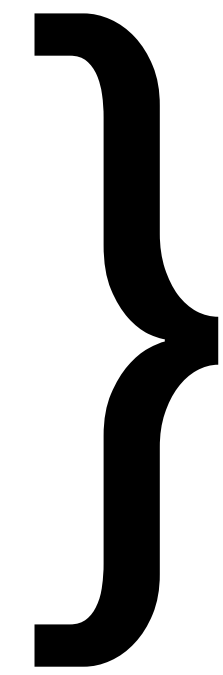
In-house Segmentation Algorithms

In-house Segmentation Algorithms

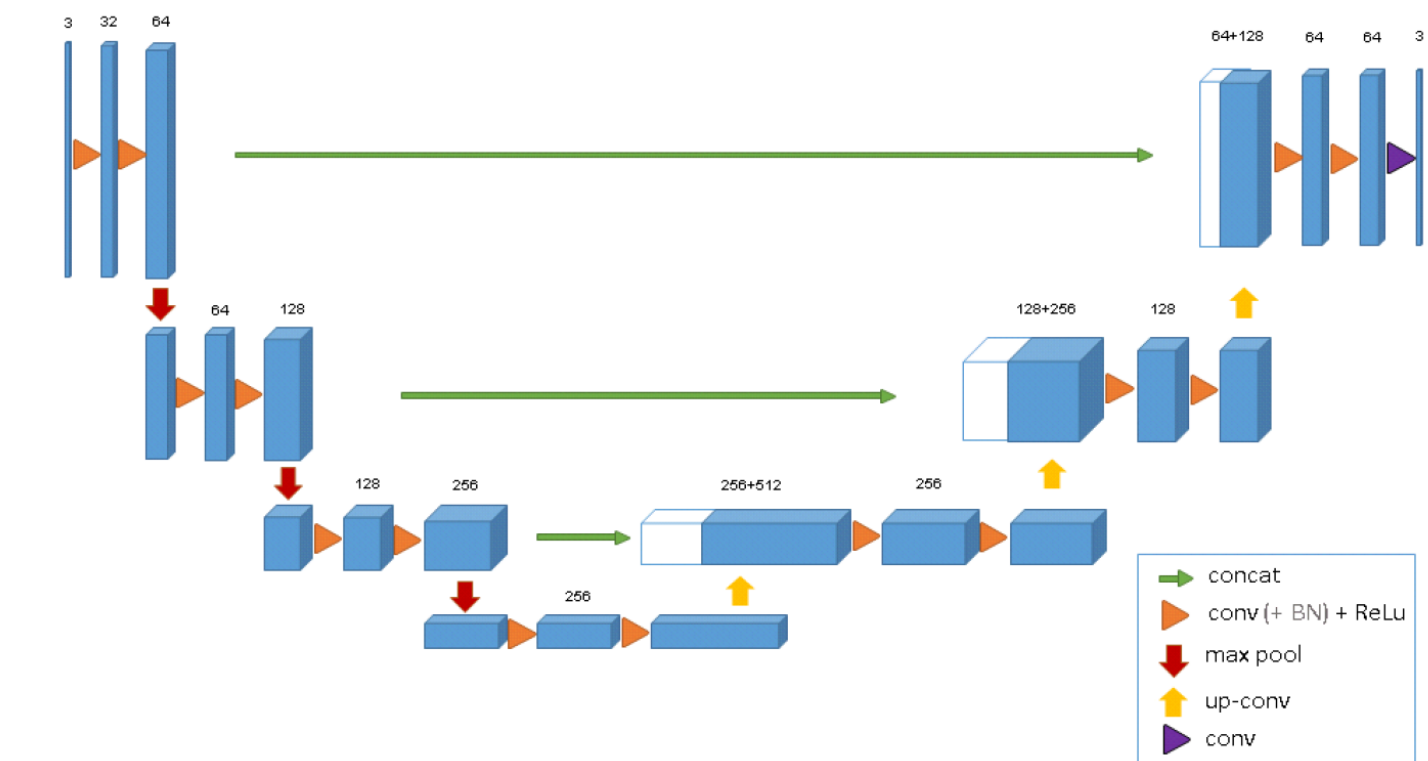
- 3D U-Net
- 3D Residual U-Net
- FCN
- Deep Medic (Pre-trained weights from the cluster)

In-house Segmentation Algorithms

- 3D U-Net
- 3D Residual U-Net
- FCN
- Deep Medic (Pre-trained weights from the cluster)

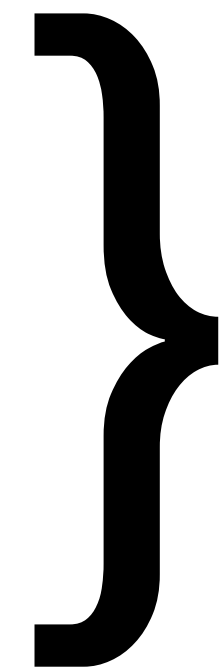


Variations of the similar
kind of U-Net
Architecture

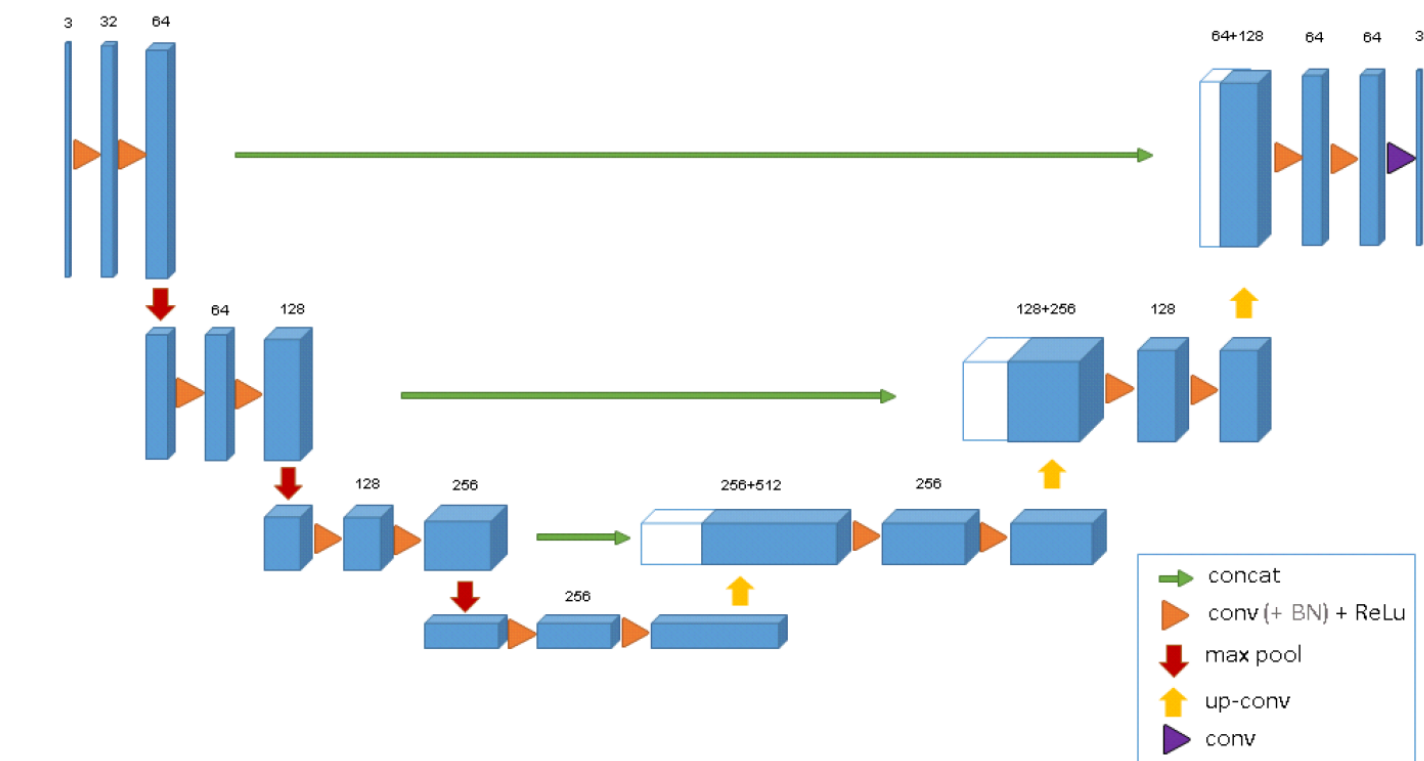


In-house Segmentation Algorithms

- 3D U-Net
- 3D Residual U-Net
- FCN



Variations of the similar
kind of U-Net
Architecture



- Deep Medic (Pre-trained weights from the cluster)
- Inception (Still in experimental phase)

BraTS Segmentation Algorithms



BraTS Segmentation Algorithms

- Algorithms from BraTS-2018 challenge (Names given below are the team names):
 - MIC-DKFZ (TU Munich)
 - PVG-18 (McGill University)
 - GBMNET-18 (University of Washington)



Where were the BraTS-18 Algorithms obtained from?



<https://hub.docker.com>

12 Algorithms in different frameworks



Caffe



theano



6 algorithms in

PYTORCH

5 Algorithms working inside their respective dockers

1 didn't work due to a bug in the inference script

```
Traceback (most recent call last):
  File "DeepSCAN_BRATS.py", line 914, in <module>
    mask,5, axis=0)
  File "DeepSCAN_BRATS.py", line 537, in apply_to_case
    outputs, logit_flip = model(images[0])
  File "/cbica/home/bhaleram/.conda/envs/mckinley-scan18/lib/python3.6/site-packages/torch/nn/modules/module.py", line 489, in __call__
    result = self.forward(*input, **kwargs)
  File "DeepSCAN_BRATS.py", line 301, in forward
    out = self.depth_reducing_layers[i](out)
  File "/cbica/home/bhaleram/.conda/envs/mckinley-scan18/lib/python3.6/site-packages/torch/nn/modules/module.py", line 489, in __call__
    result = self.forward(*input, **kwargs)
  File "/cbica/home/bhaleram/.conda/envs/mckinley-scan18/lib/python3.6/site-packages/torch/nn/modules/container.py", line 92, in forward
    input = module(input)
  File "/cbica/home/bhaleram/.conda/envs/mckinley-scan18/lib/python3.6/site-packages/torch/nn/modules/module.py", line 489, in __call__
    result = self.forward(*input, **kwargs)
  File "/cbica/home/bhaleram/.conda/envs/mckinley-scan18/lib/python3.6/site-packages/torch/nn/modules/padding.py", line 276, in forward
    return F.pad(input, self.padding, 'replicate')
  File "/cbica/home/bhaleram/.conda/envs/mckinley-scan18/lib/python3.6/site-packages/torch/nn/functional.py", line 2683, in pad
    assert len(pad) == 4, '4D tensors expect 4 values for padding'
AssertionError: 4D tensors expect 4 values for padding
```

**3 working algorithms (outside
the **docker**) - one didn't work
due to a permission issue and
the other due to an
environment issue**

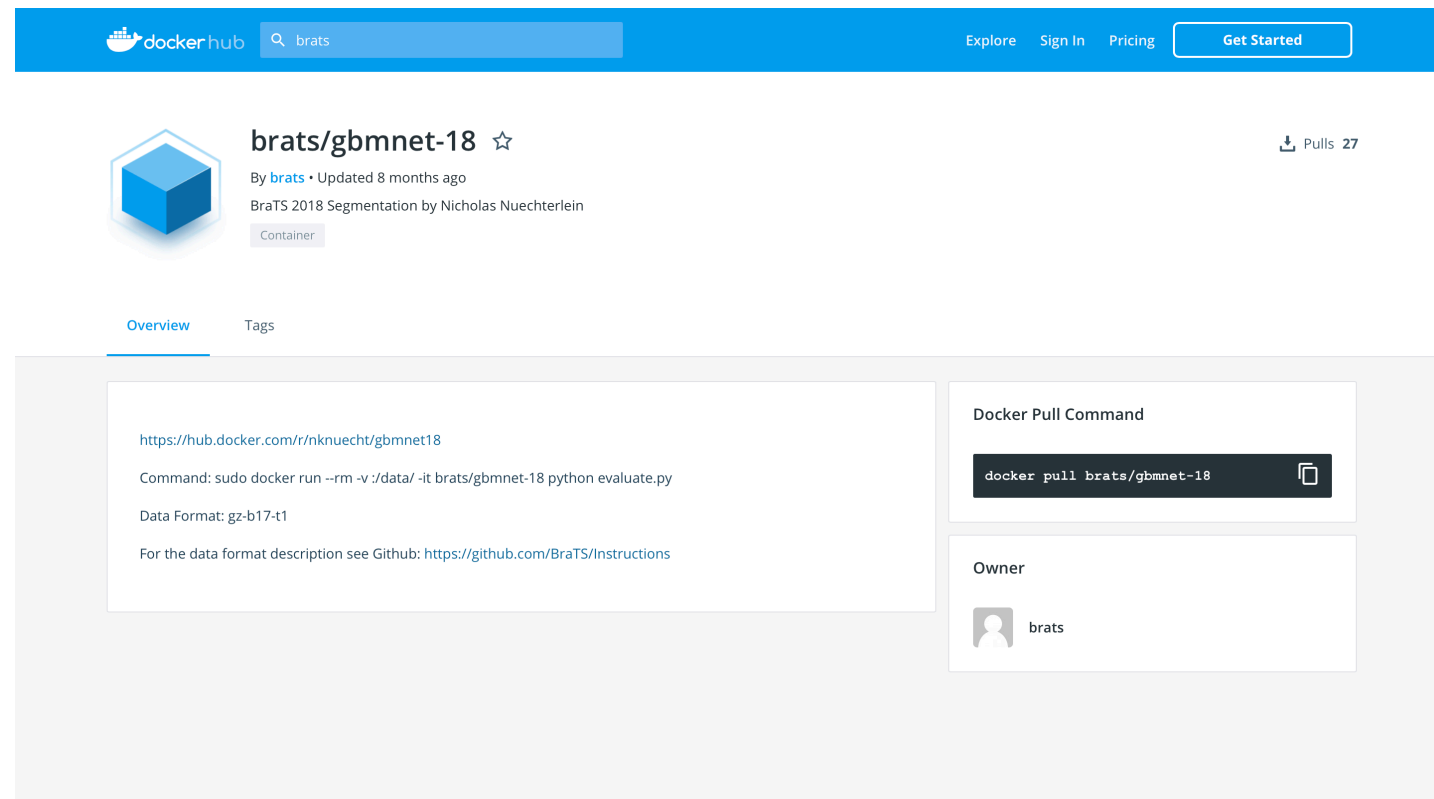
Thus, we end up with.....

- MIC-DKFZ (TU Munich)
- PVG-18 (McGill University)
- GBMNET-18 (University of Washington)



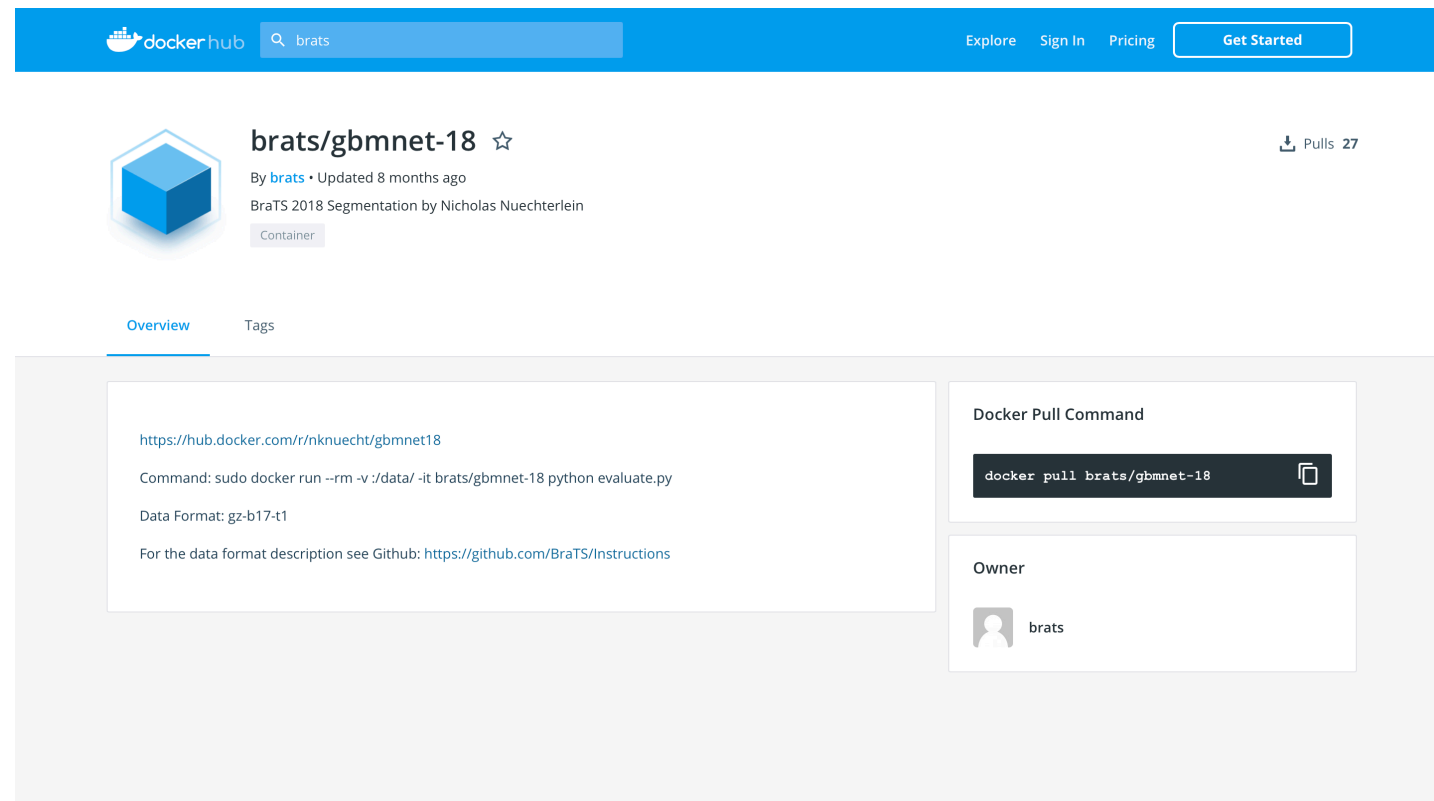
How were the 3 BraTS-18 algorithms used?

How were the 3 BraTS-18 algorithms used?



Pull (Download) the docker container on your local machine as given in the picture

How were the 3 BraTS-18 algorithms used?

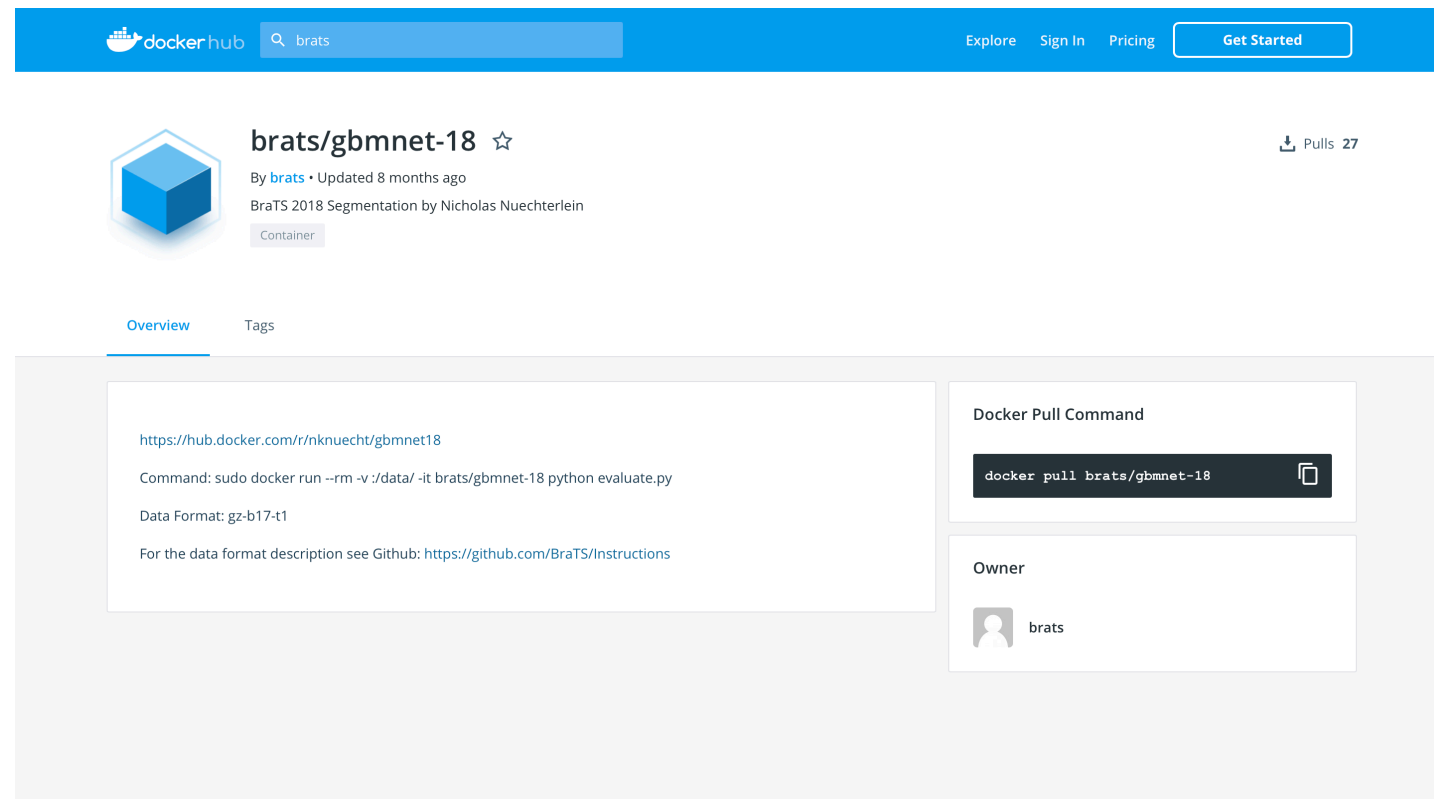


Pull (Download) the docker container on your local machine as given in the picture

Run docker on local machine



How were the 3 BraTS-18 algorithms used?



Pull (Download) the docker container on your local machine as given in the picture

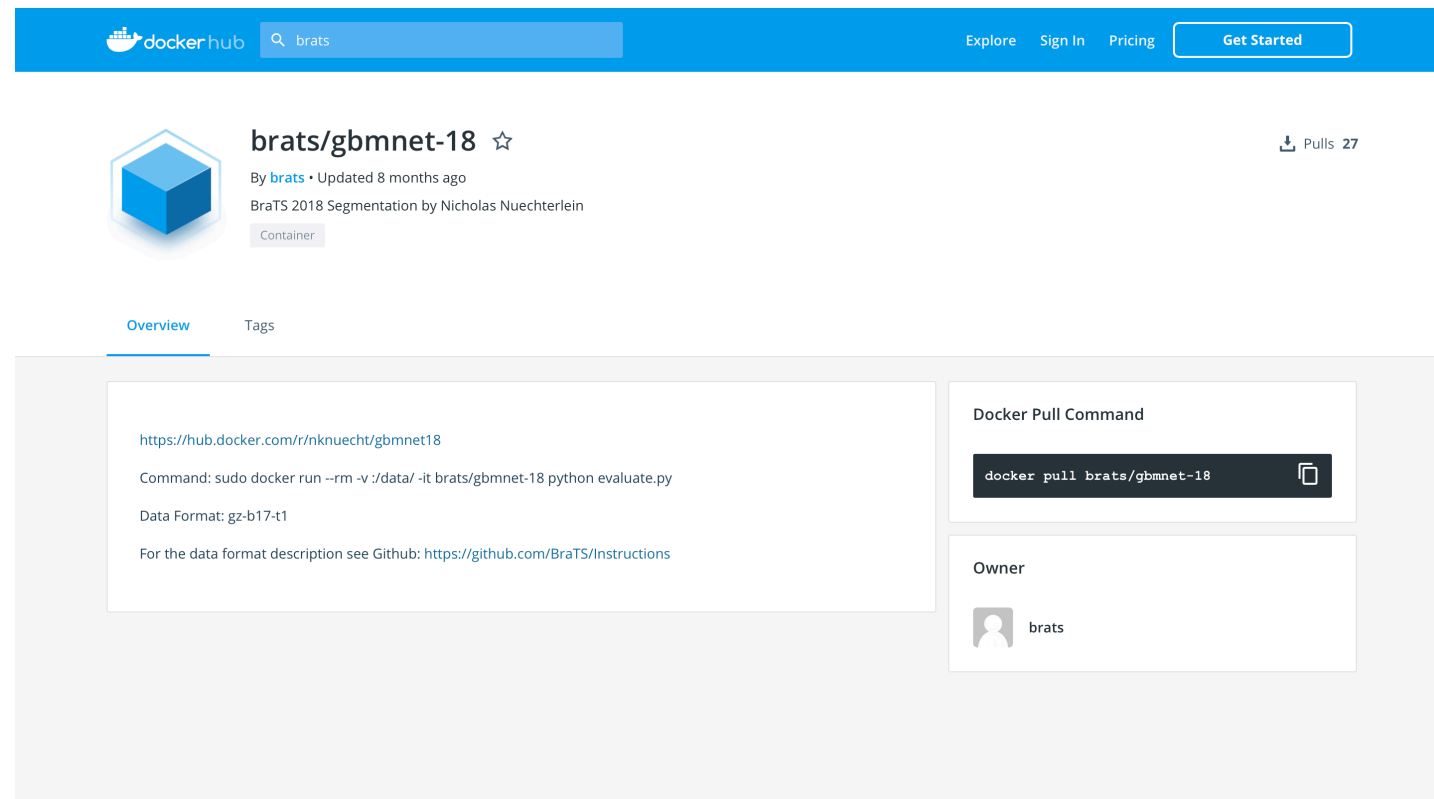
Run docker on local machine



```
megh@megh-Legion-T730-281C0:/S$ ls
bin  boot  cdrom  dev  etc  home  initrd.img  initrd.img.old  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  snap  srv  sys  usr  var  vmlinuz  vmlinuz.old
megh@megh-Legion-T730-281C0:/S$
```

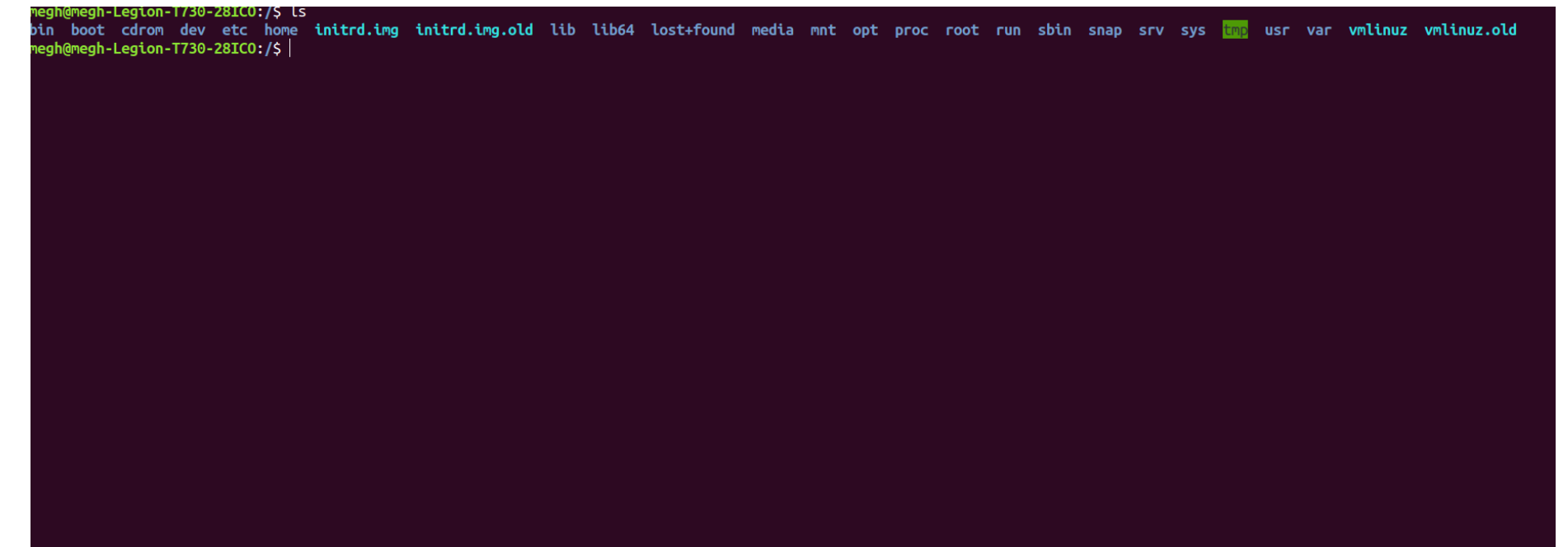
Looks like any other linux shell once we are “inside” the docker

How were the 3 BraTS-18 algorithms used?



Pull (Download) the docker container on your local machine as given in the picture

Run docker on local machine



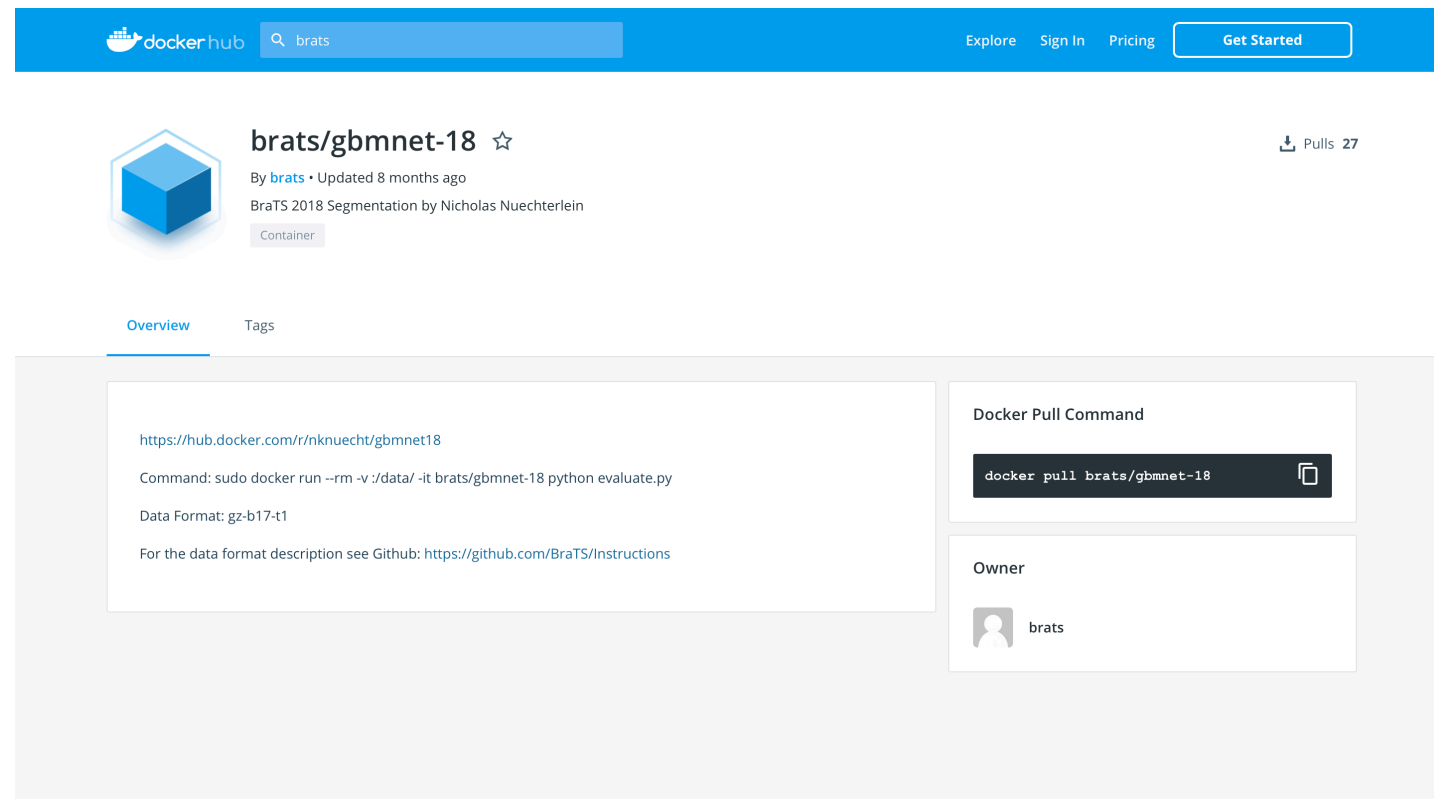
Looks like any other linux shell once we are “inside” the docker

We need only the model weights and the supporting inference scripts

**Make a list of Python Dependencies :
pip freeze /
conda list**

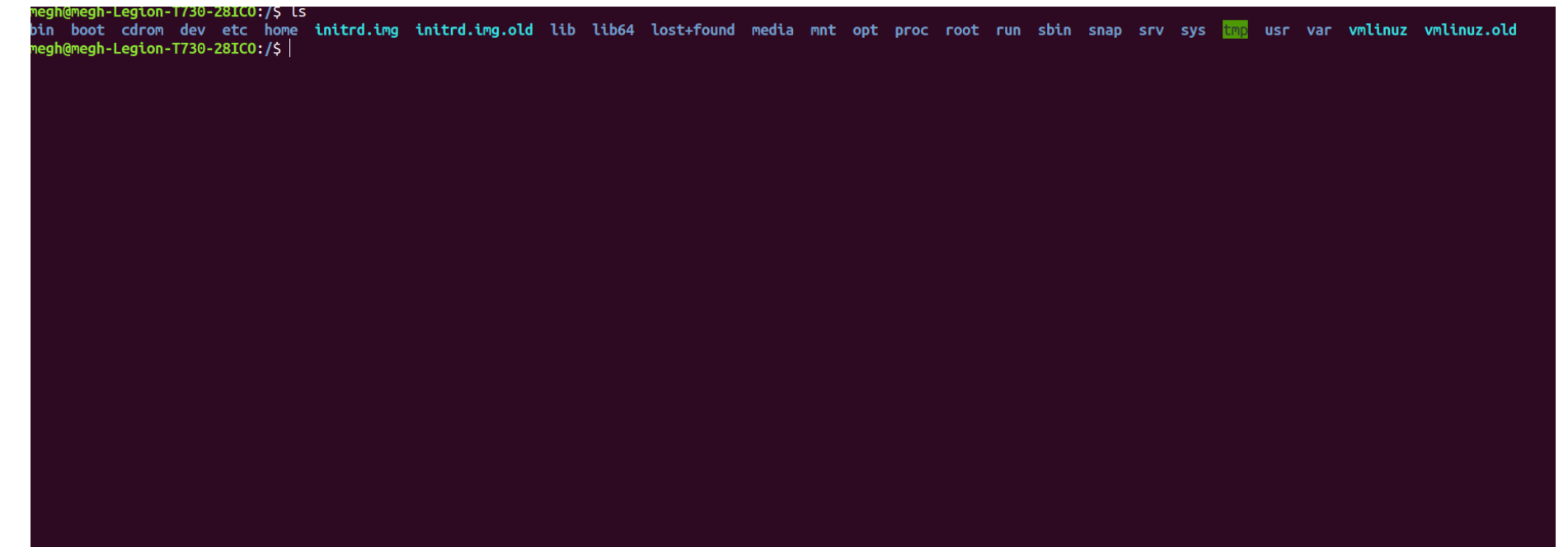


How were the 3 BraTS-18 algorithms used?



Pull (Download) the docker container on your local machine as given in the picture

Run docker on local machine



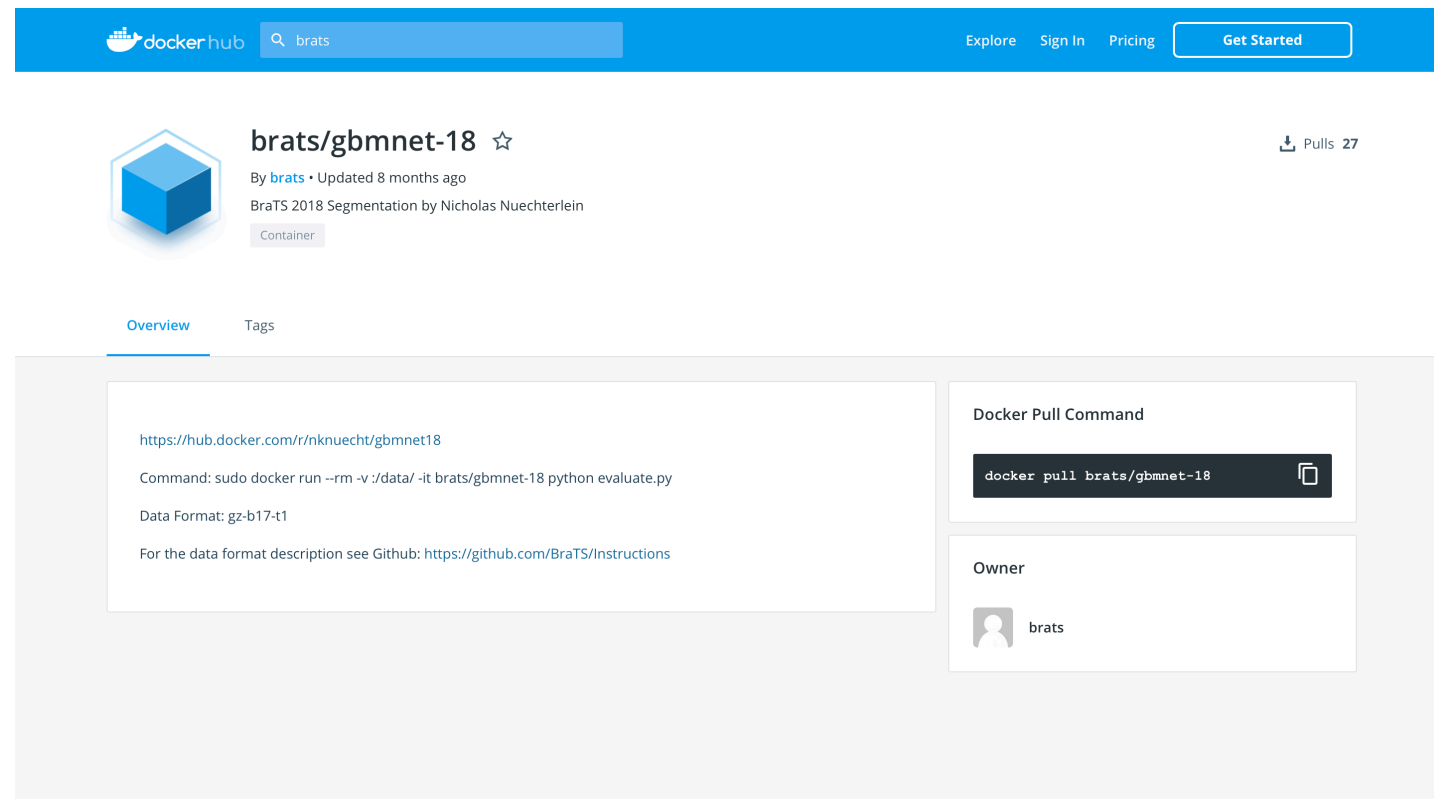
Looks like any other linux shell once we are “inside” the docker

We need only the model weights and the supporting inference scripts

**Make a list of Python Dependencies :
pip freeze /
conda list**

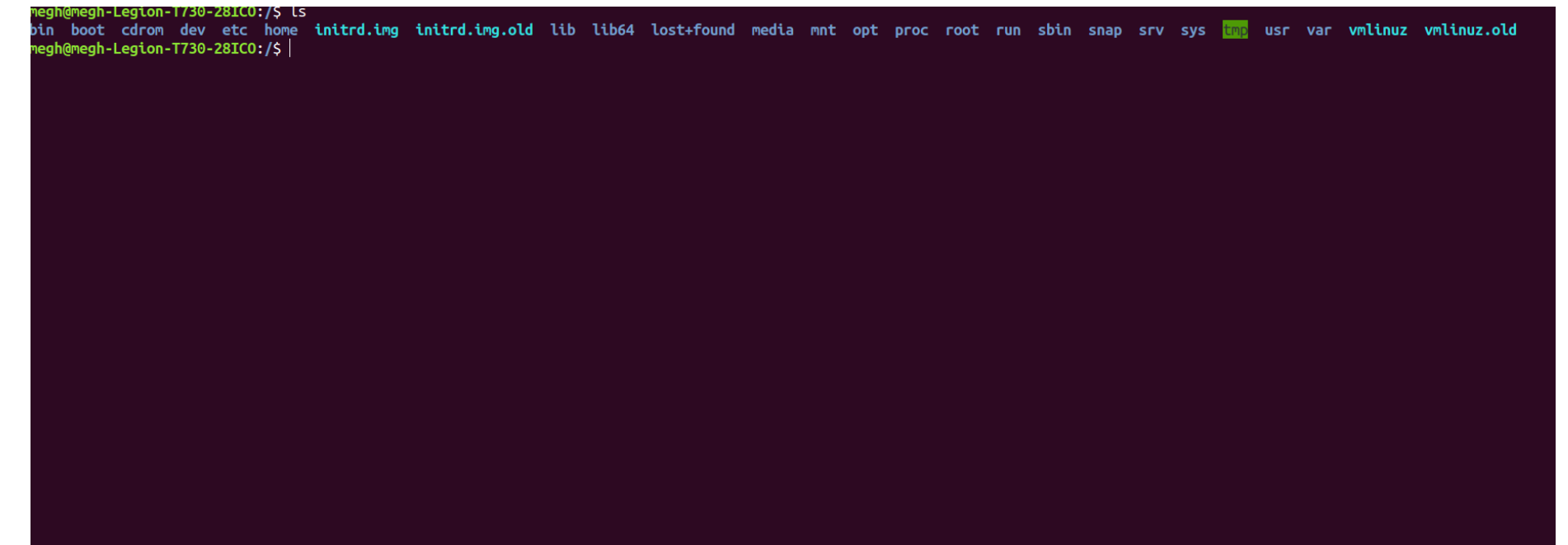
```
pillow==6.1.0
prompt-toolkit==2.0.9
psutil==5.6.3
ptyprocess==0.6.0
pycodestyle==2.5.0
pycosat==0.6.3
pycparser==2.19
pyflakes==2.1.1
pygments==2.4.2
pylint==2.3.1
pyOpenSSL==19.0.0
pyParsing==2.4.2
pyrsistent==0.14.11
pysocks==1.7.0
python-dateutil==2.8.0
pytz==2019.2
PyYAML==5.1.2
pyzmq==18.1.0
QtAwesome==0.5.7
qtconsole==4.5.5
QtPy==1.9.0
requests==2.22.0
rope==0.14.0
ruamel-yaml==0.15.46
six==1.12.0
snowballstemmer==1.9.0
Sphinx==2.1.2
sphinxcontrib-applehelp==1.0.1
sphinxcontrib-devhelp==1.0.1
sphinxcontrib-htmlhelp==1.0.2
sphinxcontrib-ismath==1.0.1
sphinxcontrib-qthelp==1.0.2
sphinxcontrib-serializinghtml==1.1.3
spyder==3.3.6
spyder-kernels==0.5.1
testpath==0.4.2
torch==1.2.0
torchvision==0.4.0a0+6b959ee
tornado==6.0.3
tqdm==4.32.1
traitlets==4.3.2
urllib3==1.24.2
wcmwidth==0.1.7
webencodings==0.5.1
wrapt==1.11.2
wurlitzer==1.0.3
```


How were the 3 BraTS-18 algorithms used?



Pull (Download) the docker container on your local machine as given in the picture

Run docker on local machine



Looks like any other linux shell once we are “inside” the docker

We need only the model weights and the supporting inference scripts

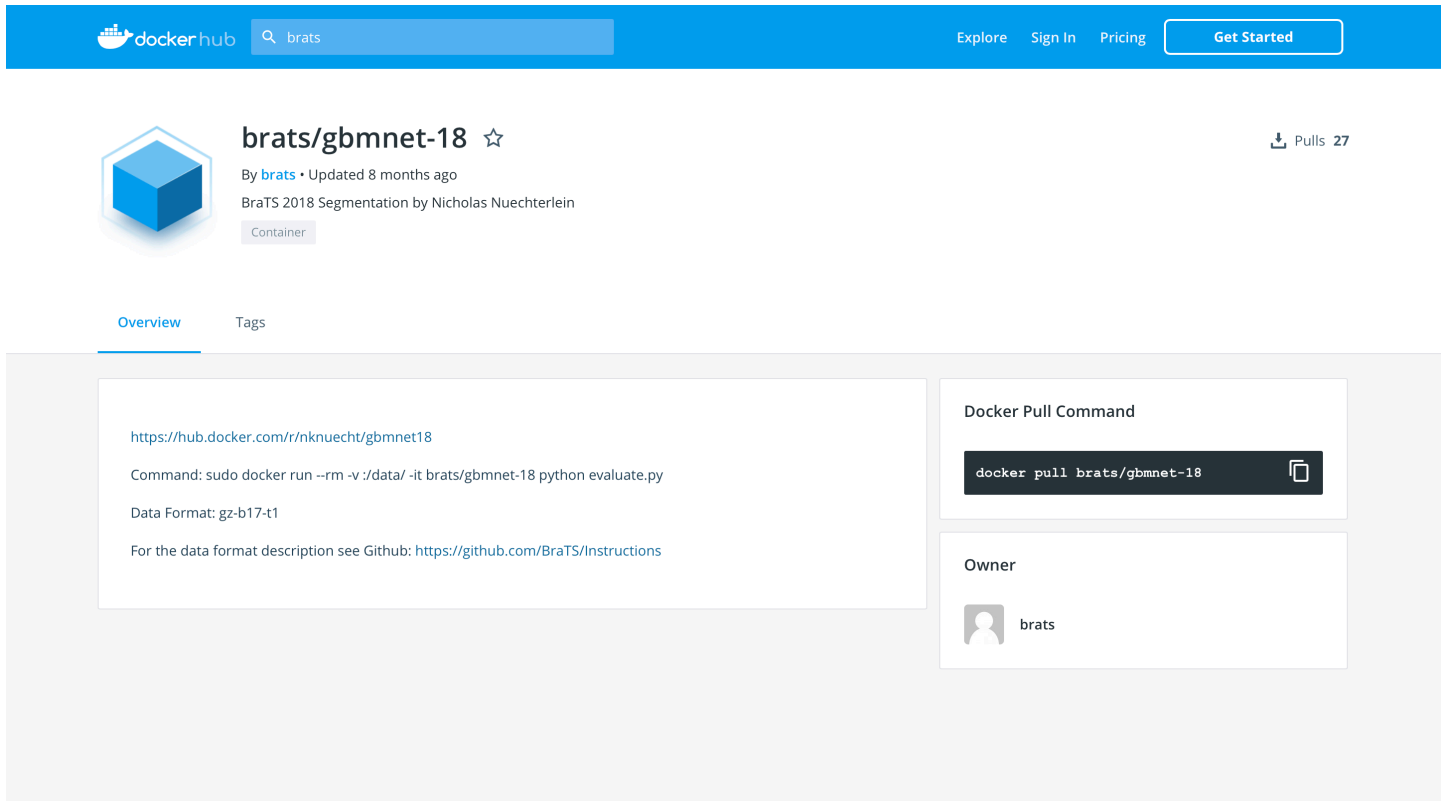
**Make a list of Python Dependencies :
pip freeze /
conda list**

Copy relevant items from “inside” the docker to local machine/cluster using the : docker cp command



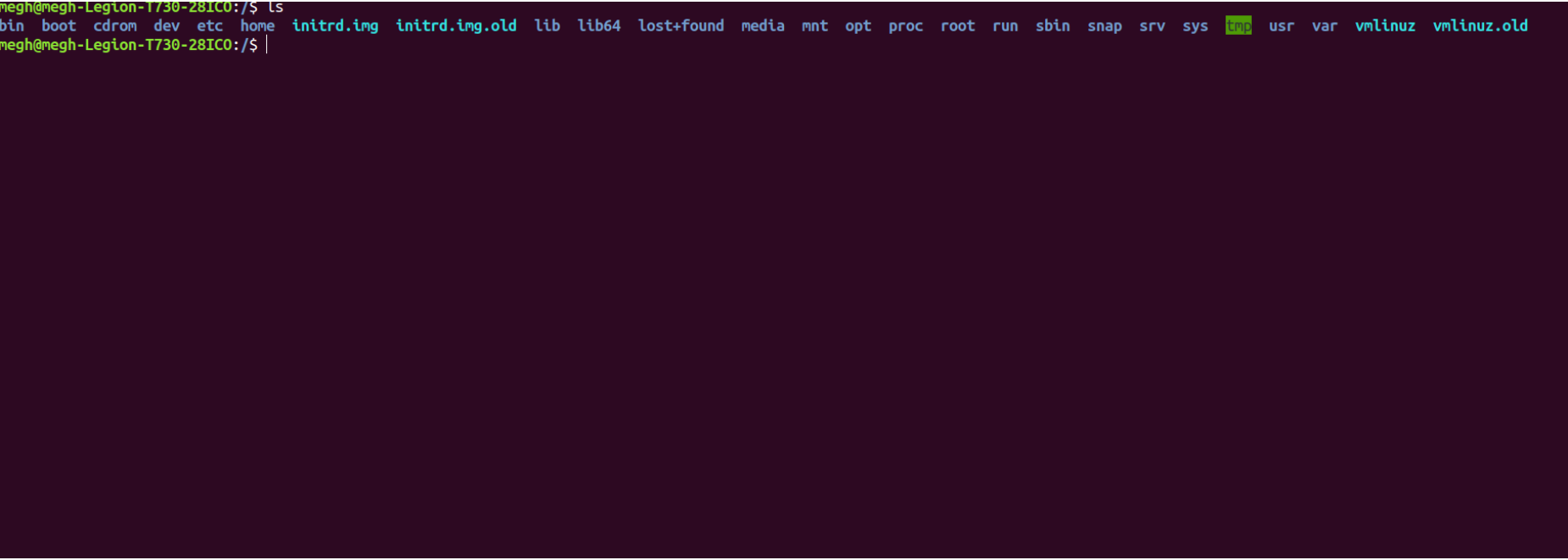
```
pillow==6.1.0
prompt-toolkit==2.0.9
psutil==5.6.3
ptyprocess==0.6.0
pycodestyle==2.5.0
pycosat==0.6.3
pycparser==2.11
pyflakes==2.1.1
pygments==2.4.2
pylint==2.3.1
pyOpenSSL==19.0.0
pyrsistent==0.14.11
pysocks==1.7.0
python-dateutil==2.8.0
pytz==2019.2
PyYAML==5.1.2
pyzmq==18.1.0
QtAwesome==0.5.7
qtconsole==4.5.5
QtPy==1.9.0
requests==2.22.0
rope==0.14.0
ruamel-yaml==0.15.46
six==1.12.0
snowballstemmer==1.9.0
Sphinx==2.1.2
sphinxcontrib-applehelp==1.0.1
sphinxcontrib-devhelp==1.0.1
sphinxcontrib-htmlhelp==1.0.2
sphinxcontrib-ismath==1.0.1
sphinxcontrib-qthelp==1.0.2
sphinxcontrib-serializinghtml==1.1.3
spyder==3.3.6
spyder-kernels==0.5.1
testpath==0.4.2
torch==1.2.0
torchvision==0.4.0a0+6b959ee
tornado==6.0.3
tqdm==4.32.1
traitlets==4.3.2
urllib3==1.24.2
wcmwidth==0.1.7
webencodings==0.5.1
wrapt==1.11.2
wurlitzer==1.0.3
```

How were the 3 BraTS-18 algorithms used?



Pull (Download) the docker container on your local machine as given in the picture

Run docker on local machine



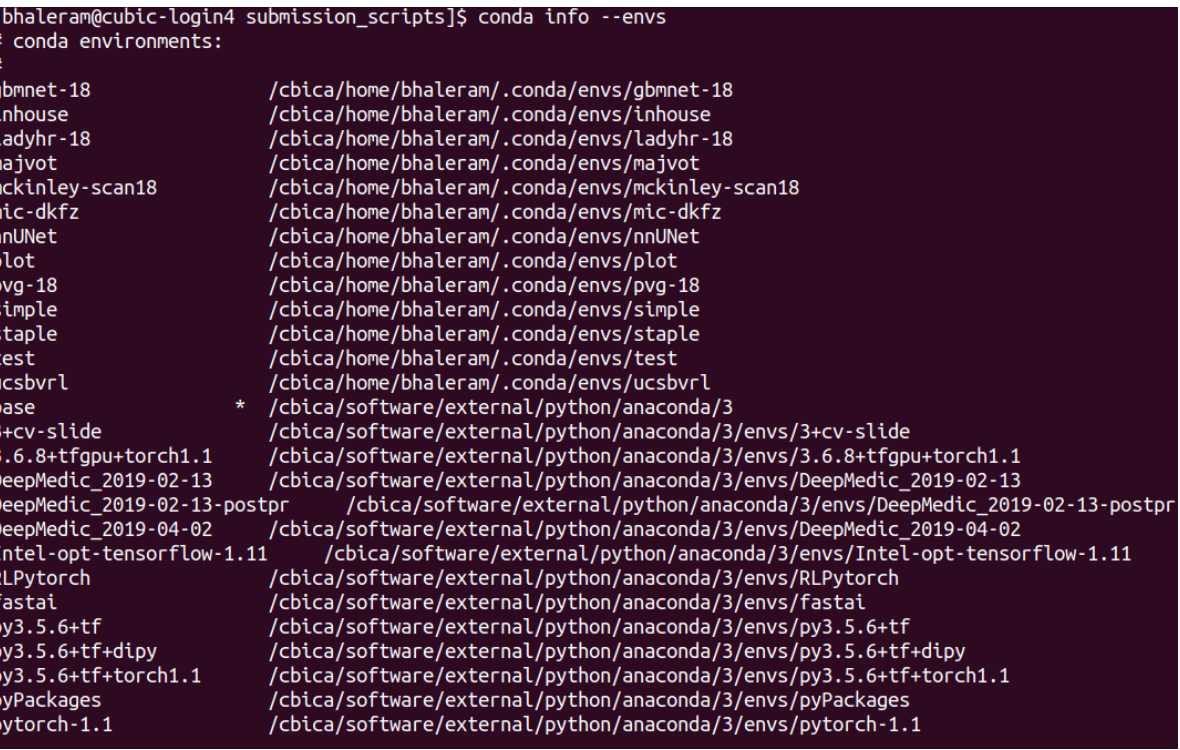
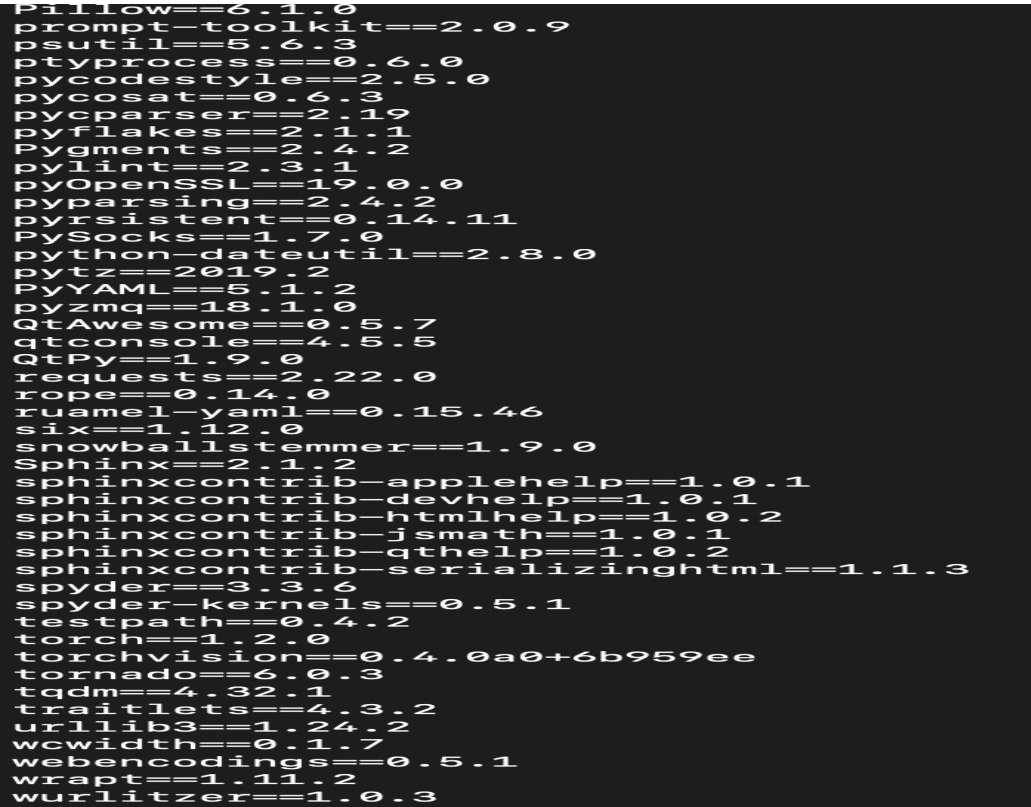
Looks like any other linux shell once we are “inside” the docker

We need only the model weights and the supporting inference scripts

**Make a list of Python Dependencies :
pip freeze /
conda list**



Copy relevant items from “inside” the docker to local machine/cluster using the : docker cp command



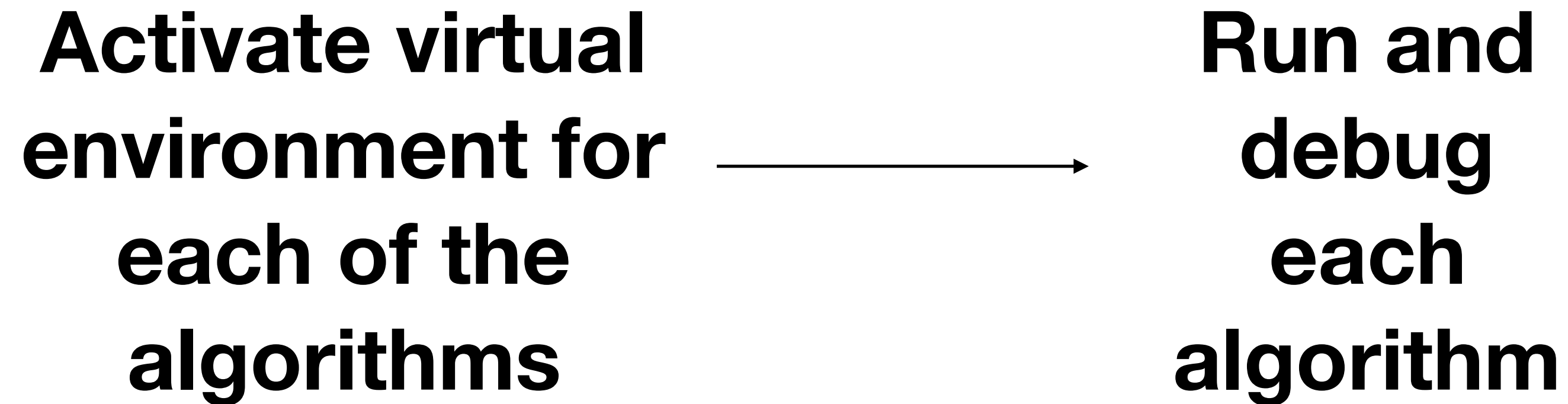
Create a virtual environment for each algorithm (including in-house) and install all the necessary dependencies

Inference using all the algorithms

Inference using all the algorithms

**Activate virtual
environment for
each of the
algorithms**

Inference using all the algorithms



Inference using all the algorithms

**Activate virtual
environment for
each of the
algorithms**

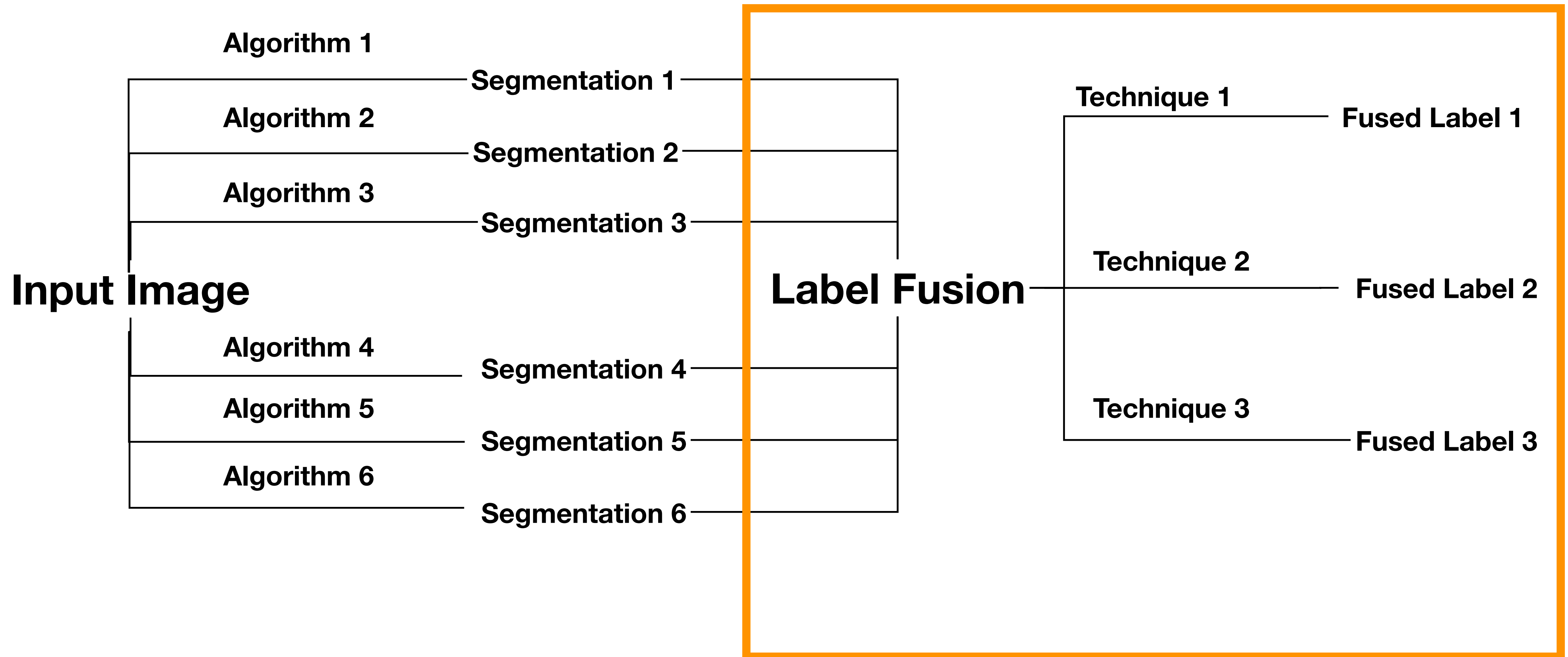


**Run and
debug
each
algorithm**



**Design inference
script so as to
accept image and
model as
command line
arguments and
output the
segmentation**

Basic FeTS Concept



Label Fusion Approaches on the generated segmentations

Label Fusion Approaches on the generated segmentations

- **Majority voting** : Voxel-Wise simple majority voting

Label Fusion Approaches on the generated segmentations

- **Majority voting** : Voxel-Wise simple majority voting
- **SIMPLE** : Selective and Iterative Method for Performance Level Estimation (T. R. Langerak, U. A. van der Heide, A. N. T. J. Kotte, M. A. Viergever, M. van Vulpen and J. P. W. Pluim, "Label Fusion in Atlas-Based Segmentation Using a Selective and Iterative Method for Performance Level Estimation (SIMPLE)," in *IEEE Transactions on Medical Imaging*, vol. 29, no. 12, pp. 2000-2008, Dec. 2010)

Label Fusion Approaches on the generated segmentations

- **Majority voting** : Voxel-Wise simple majority voting
- **SIMPLE** : Selective and Iterative Method for Performance Level Estimation (T. R. Langerak, U. A. van der Heide, A. N. T. J. Kotte, M. A. Viergever, M. van Vulpen and J. P. W. Pluim, "Label Fusion in Atlas-Based Segmentation Using a Selective and Iterative Method for Performance Level Estimation (SIMPLE)," in *IEEE Transactions on Medical Imaging*, vol. 29, no. 12, pp. 2000-2008, Dec. 2010)
- **STAPLE** : Simultaneous Truth and Performance Level Estimation (Warfield SK, Zou KH, Wells WM. Simultaneous truth and performance level estimation (STAPLE): an algorithm for the validation of image segmentation. *IEEE Trans Med Imaging*. 2004;23(7): 903–921. doi:10.1109/TMI.2004.828354)

SIMPLE Label Fusion steps

SIMPLE Label Fusion steps

- Initial segmentation generated using voxel-wise majority voting

SIMPLE Label Fusion steps

- Initial segmentation generated using voxel-wise majority voting
- Each individual segmentation is compared to the initial segmentation and performance metric is obtained.

SIMPLE Label Fusion steps

- Initial segmentation generated using voxel-wise majority voting
- Each individual segmentation is compared to the initial segmentation and performance metric is obtained.
- New consensus segmentation estimated using weighted majority voting with prior performance as weights.

SIMPLE Label Fusion steps

- Initial segmentation generated using voxel-wise majority voting
- Each individual segmentation is compared to the initial segmentation and performance metric is obtained.
- New consensus segmentation estimated using weighted majority voting with prior performance as weights.
- Worst performing segmentation is discarded and the two above steps are repeated till getting a final consensus segmentation

STAPLE Label Fusion steps

STAPLE Label Fusion steps

- Expectation-Maximization Algorithm

<https://pypi.org/project/staple/>

STAPLE Label Fusion steps

- Expectation-Maximization Algorithm
- Initial Assumption of Sensitivity (p) and Specificity (q) for each segmentation

<https://pypi.org/project/staple/>

STAPLE Label Fusion steps

- Expectation-Maximization Algorithm
- Initial Assumption of Sensitivity (p) and Specificity (q) for each segmentation
- Expectation Step (E): Calculation of consensus segmentation based on p, q and segmentations

<https://pypi.org/project/staple/>

STAPLE Label Fusion steps

- Expectation-Maximization Algorithm
- Initial Assumption of Sensitivity (p) and Specificity (q) for each segmentation
- Expectation Step (E): Calculation of consensus segmentation based on p, q and segmentations
- Maximization Step (M): Calculation of p and q based on this consensus segmentation

<https://pypi.org/project/staple/>

STAPLE Label Fusion steps

- Expectation-Maximization Algorithm
- Initial Assumption of Sensitivity (p) and Specificity (q) for each segmentation
- Expectation Step (E): Calculation of consensus segmentation based on p, q and segmentations
- Maximization Step (M): Calculation of p and q based on this consensus segmentation
- Repeat Steps E and M till convergence (convergence calculated based on absolute diff b/w 2 subsequent predictions)

<https://pypi.org/project/staple/>

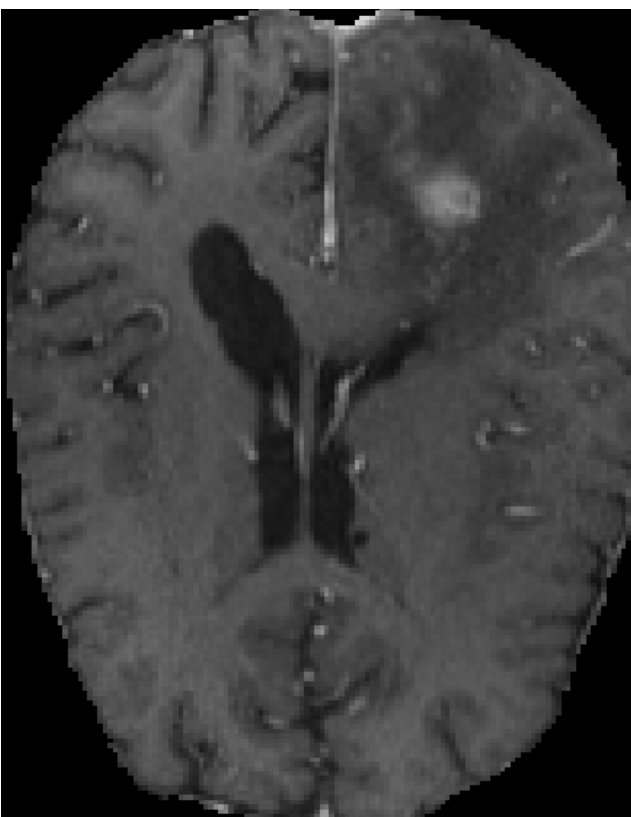
Some Qualitative Results.....

Inputs

Flair



T1ce



Inputs

DeepMedic

Flair



T1ce

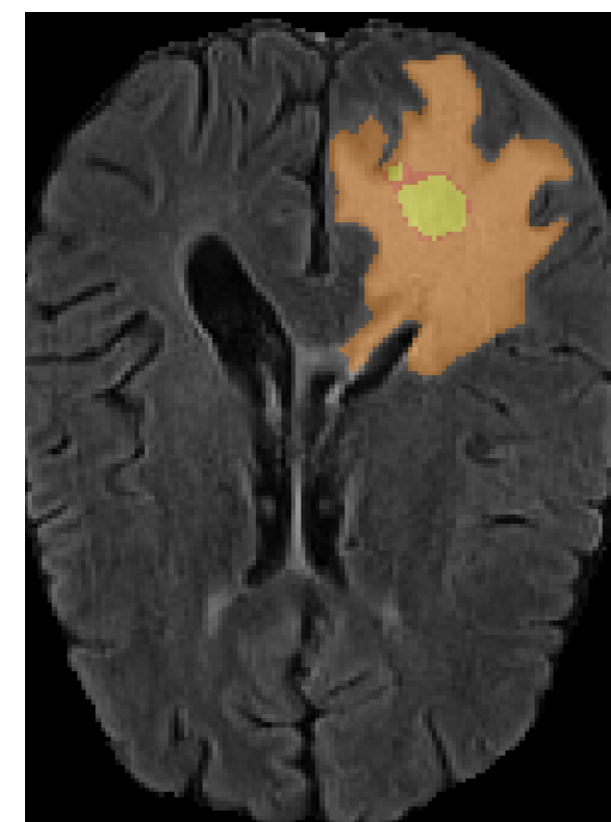


Inputs

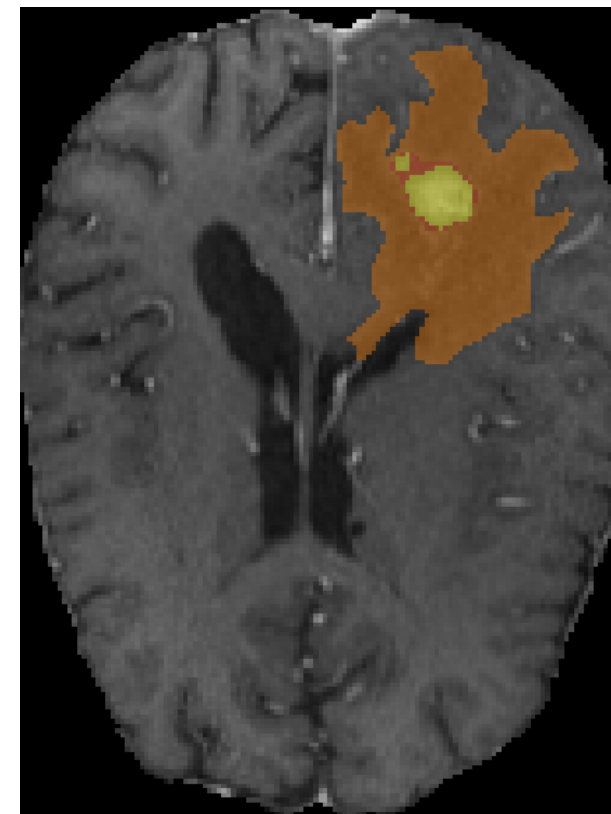
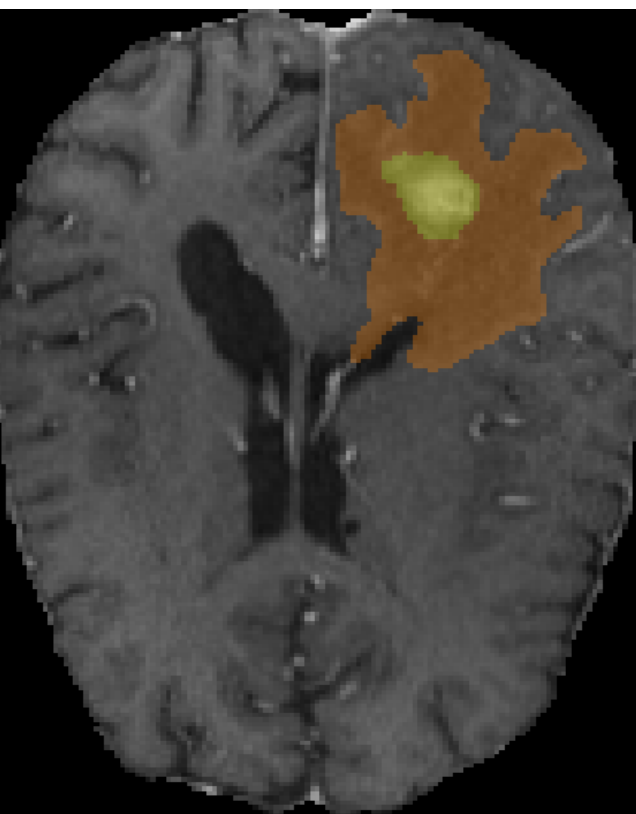
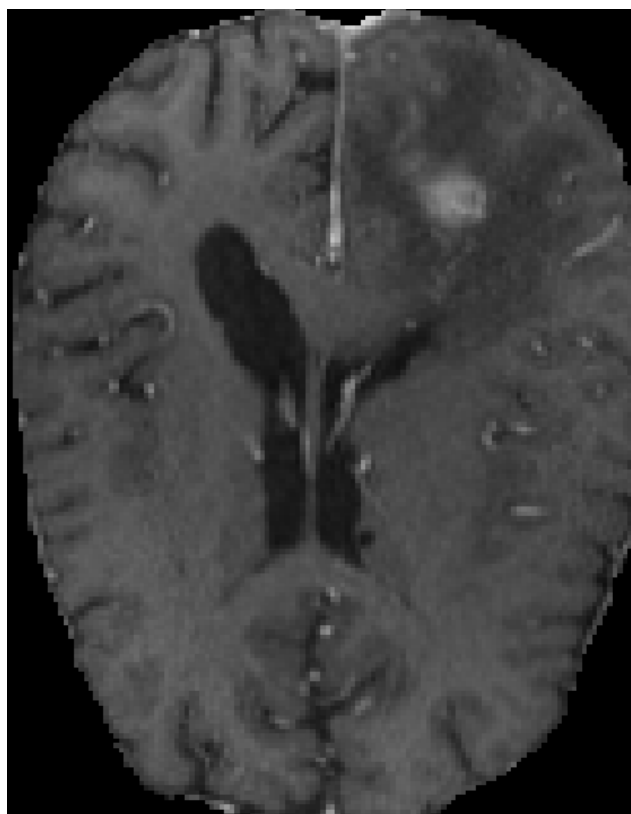
DeepMedic

DKFZ

Flair



T1ce



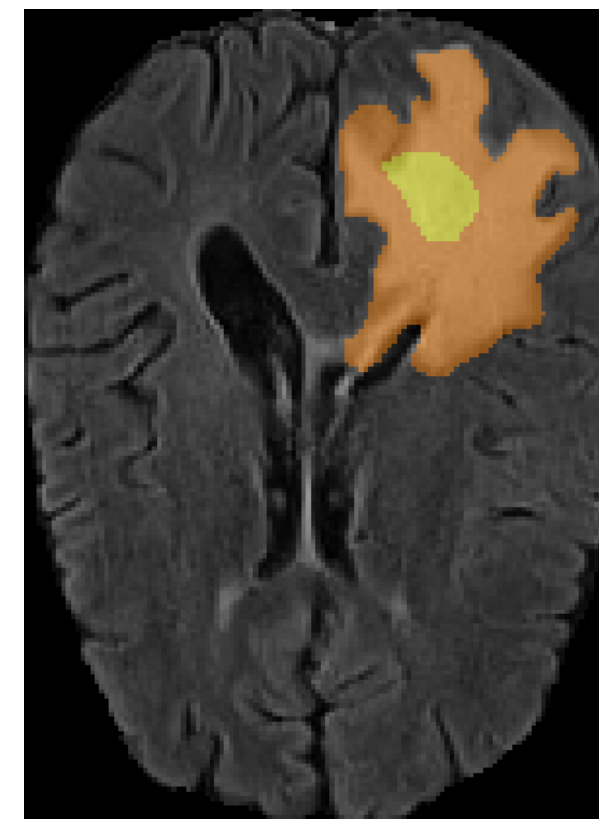
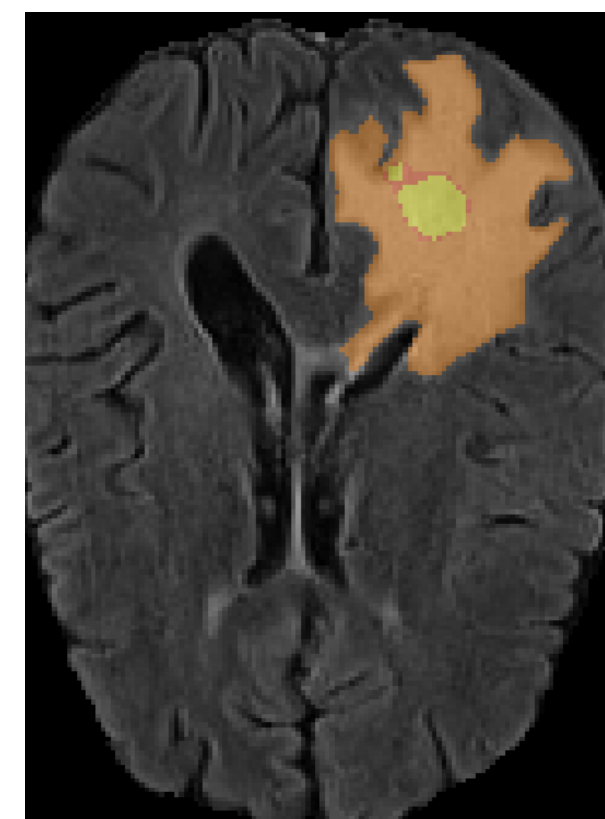
Inputs

DeepMedic

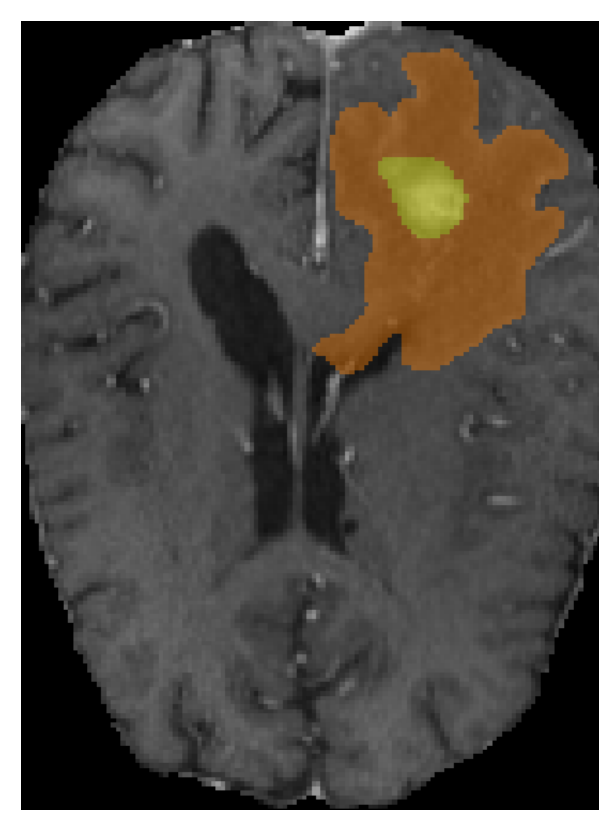
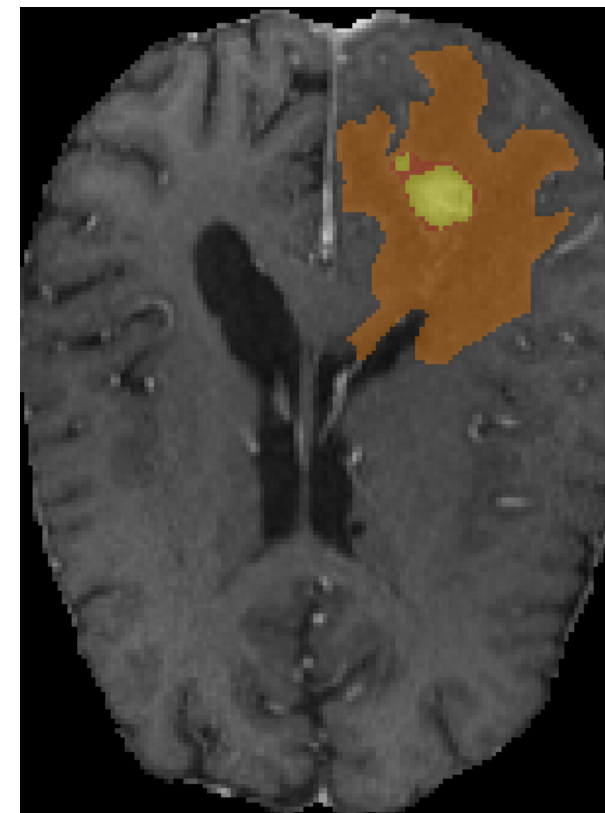
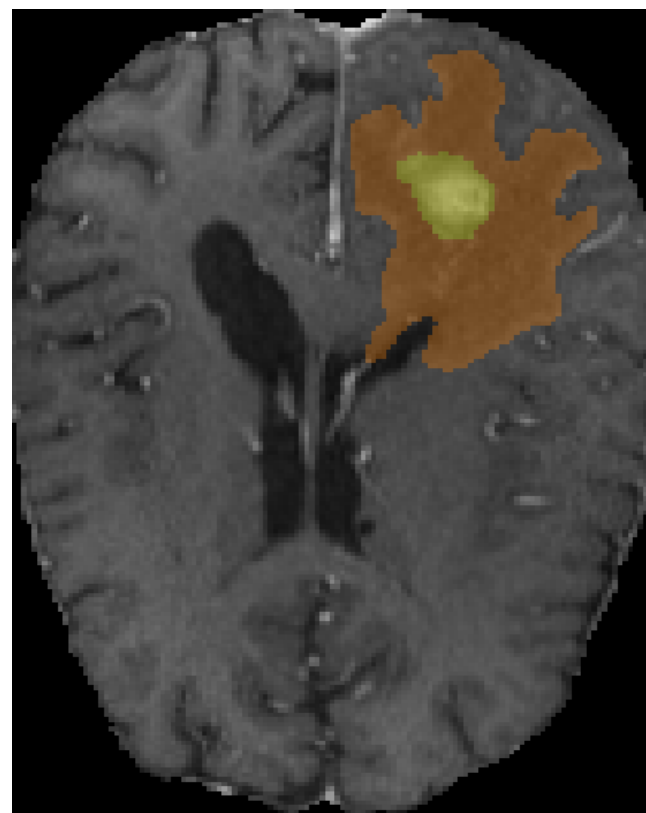
DKFZ

GBMNet18

Flair



T1ce



Inputs

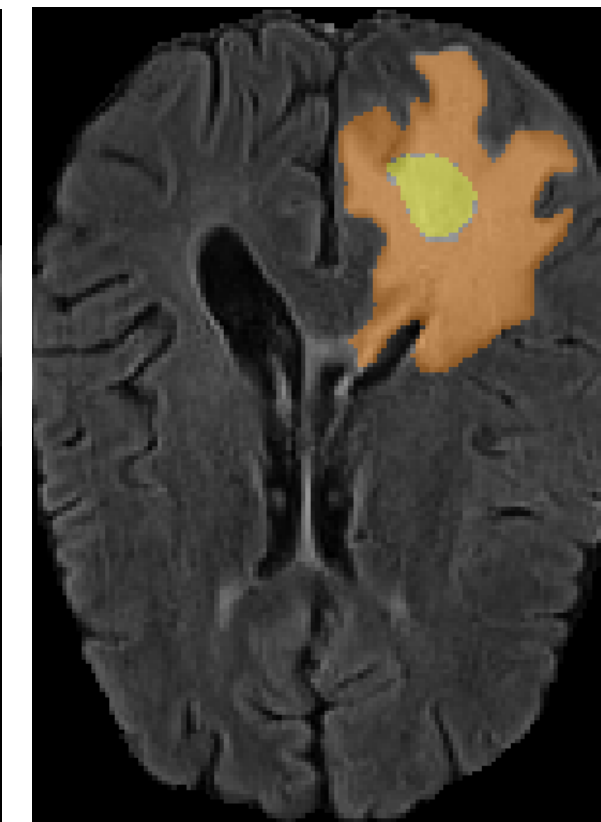
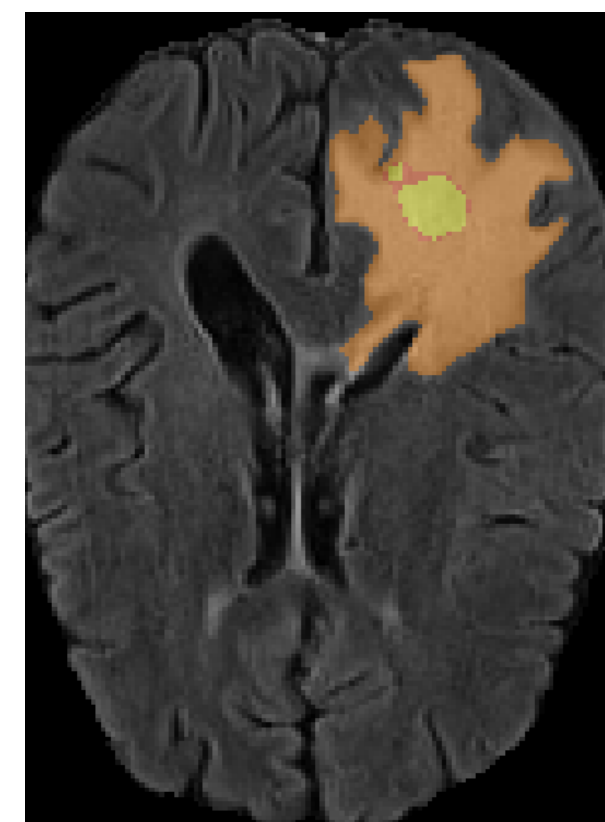
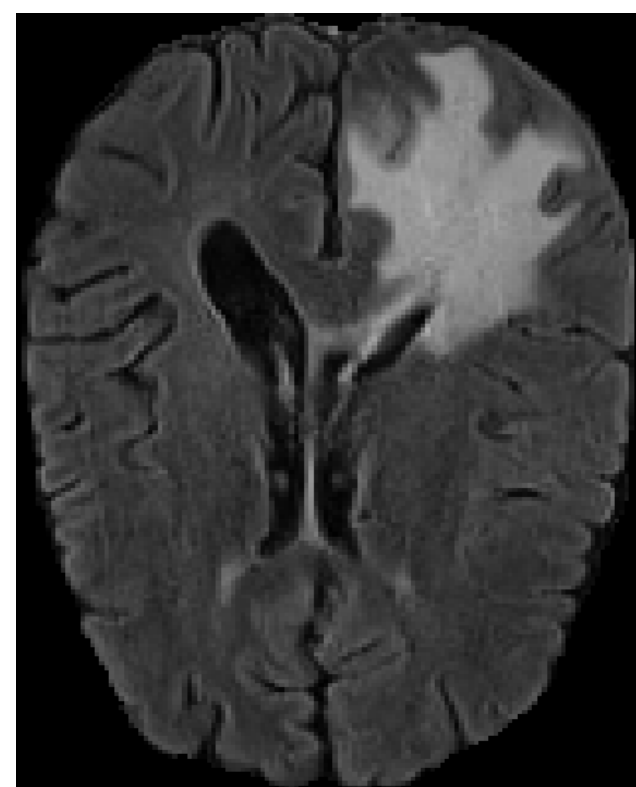
DeepMedic

DKFZ

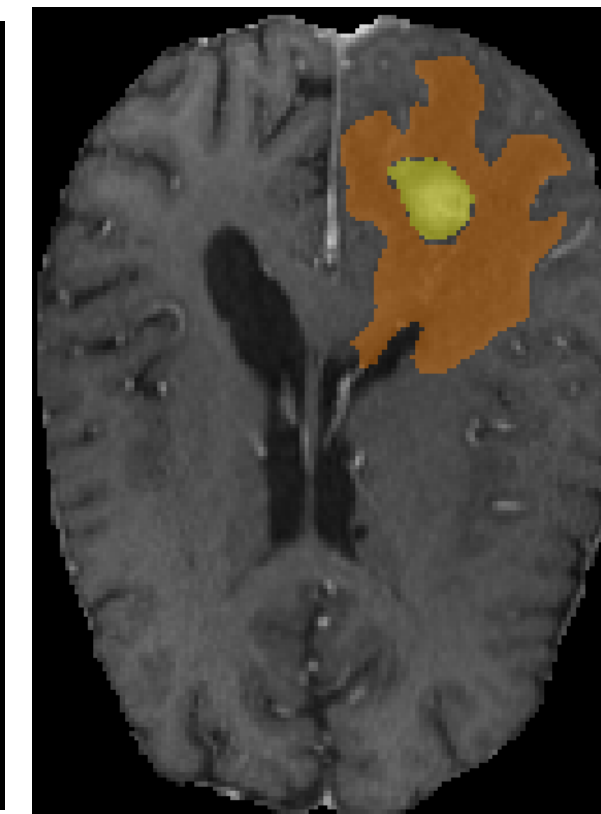
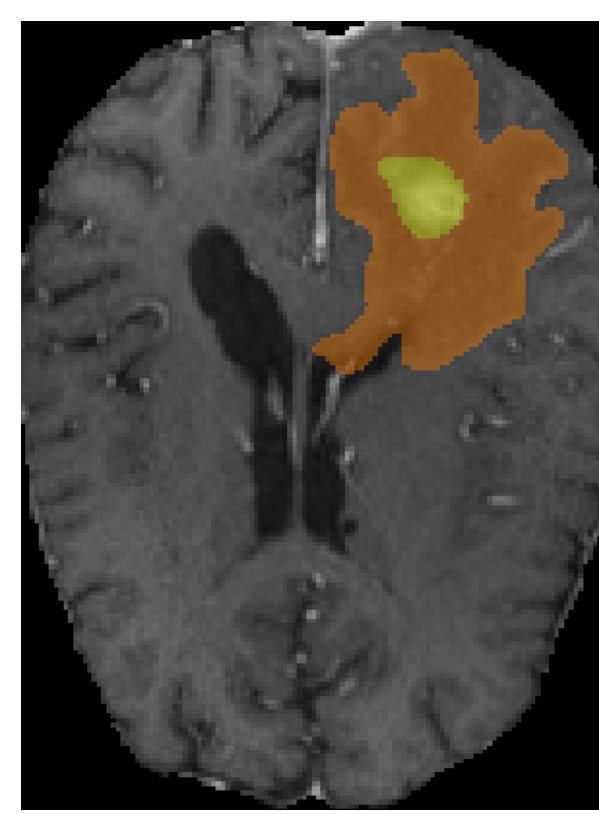
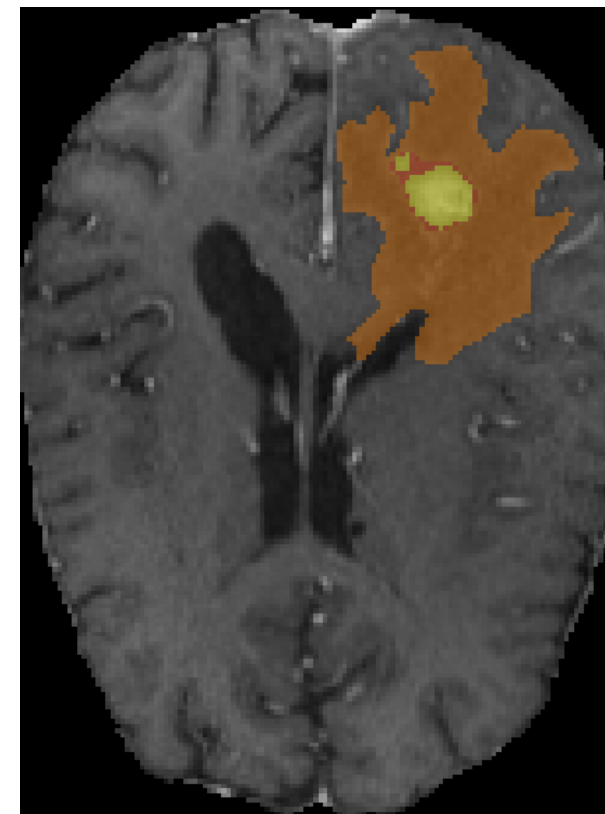
GBMNet18

Majority Voting

Flair



T1ce



Inputs

DeepMedic

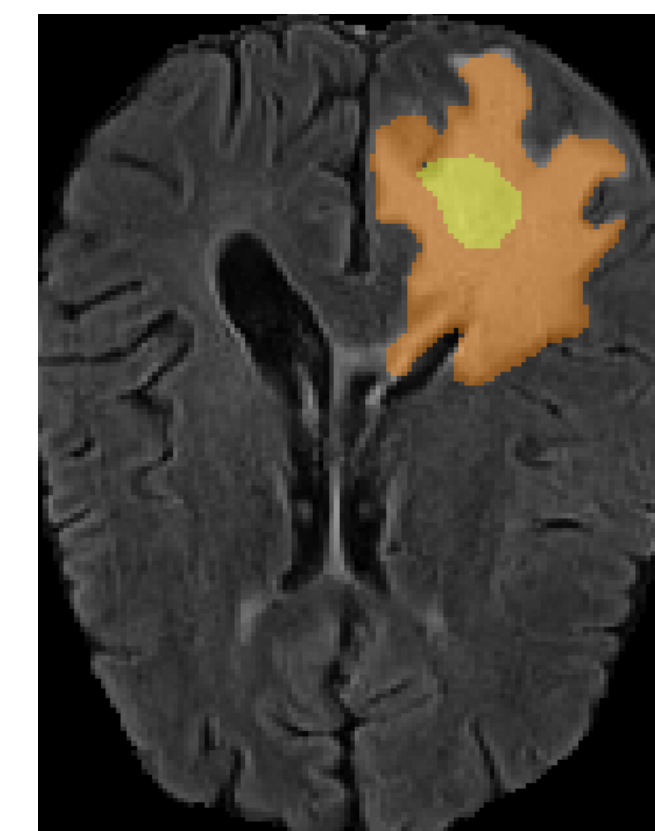
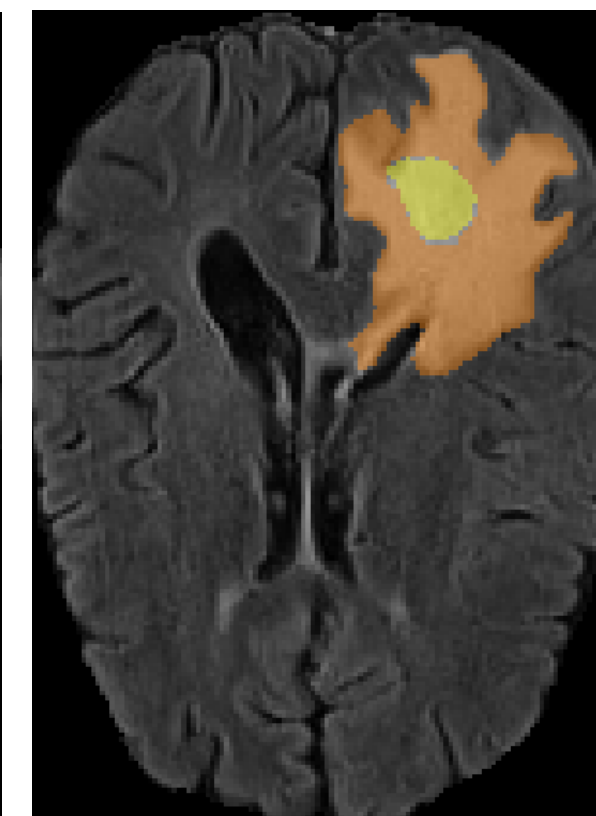
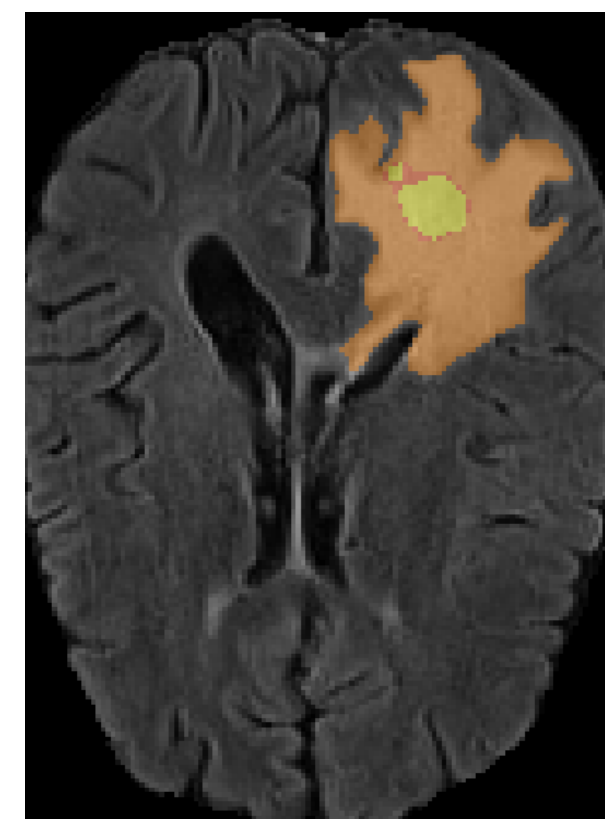
DKFZ

GBMNet18

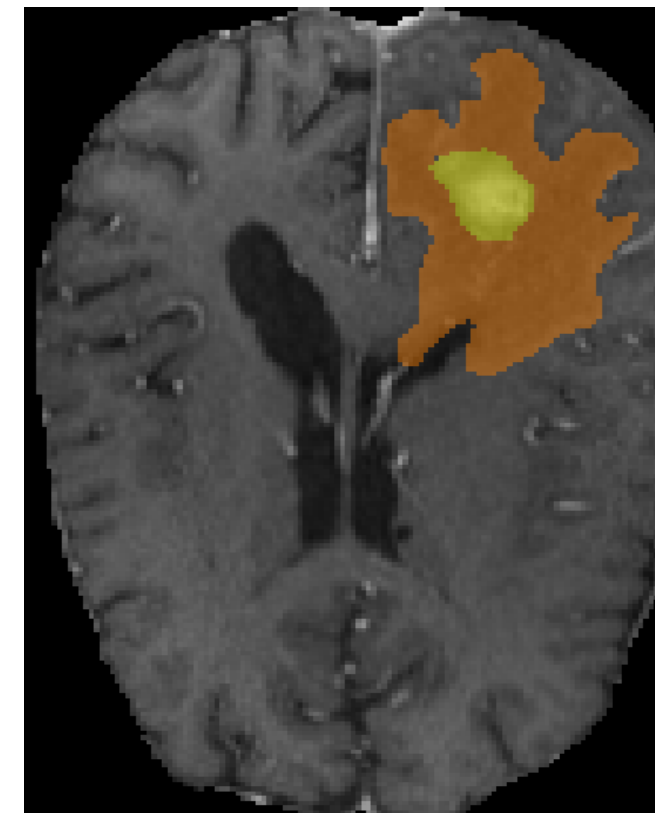
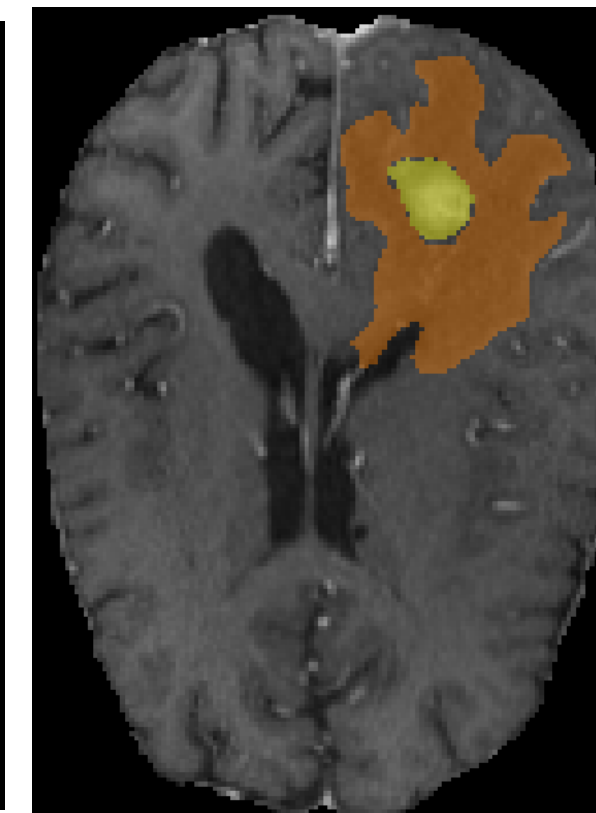
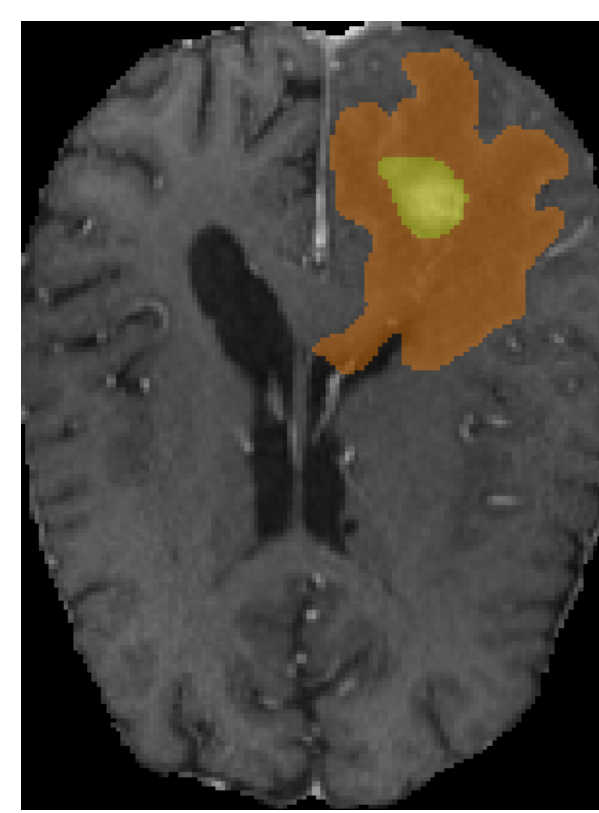
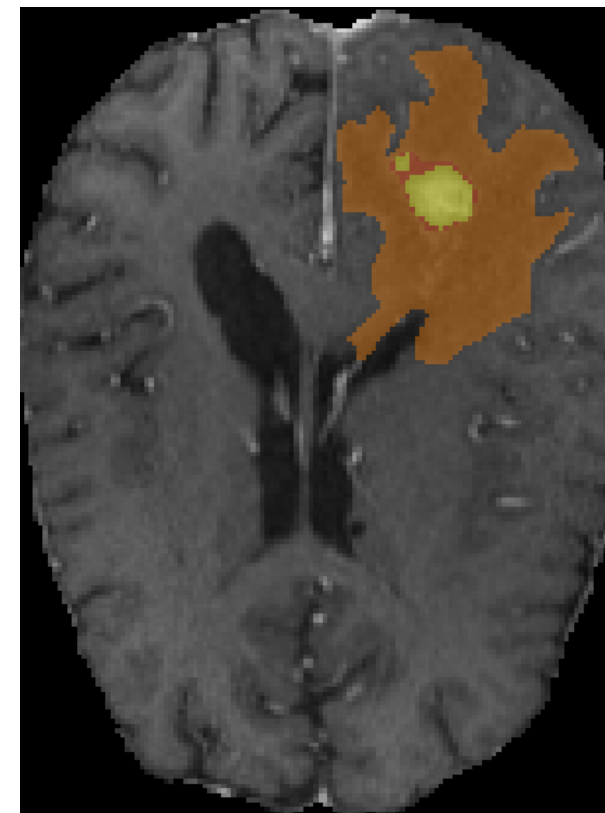
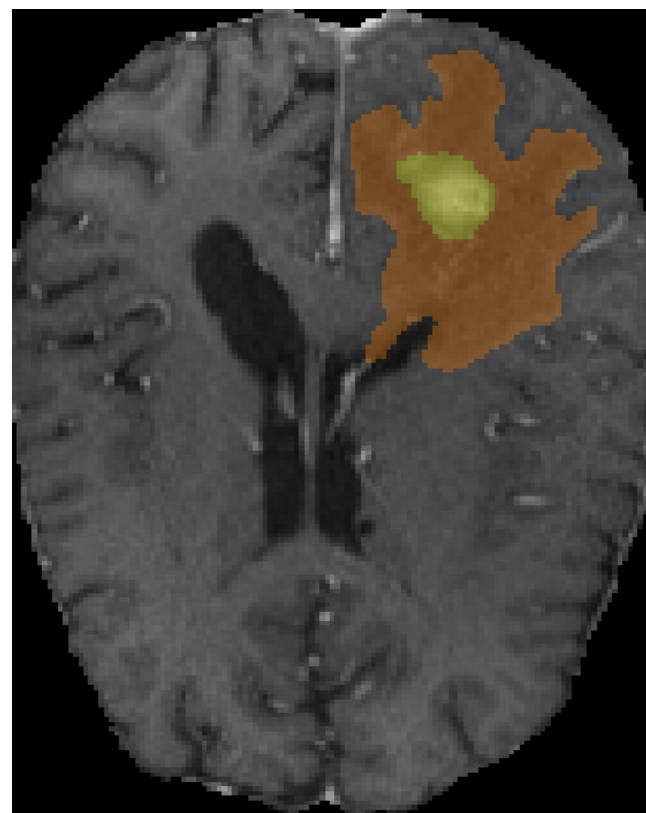
Majority Voting

SIMPLE

Flair



T1ce



Inputs

DeepMedic

DKFZ

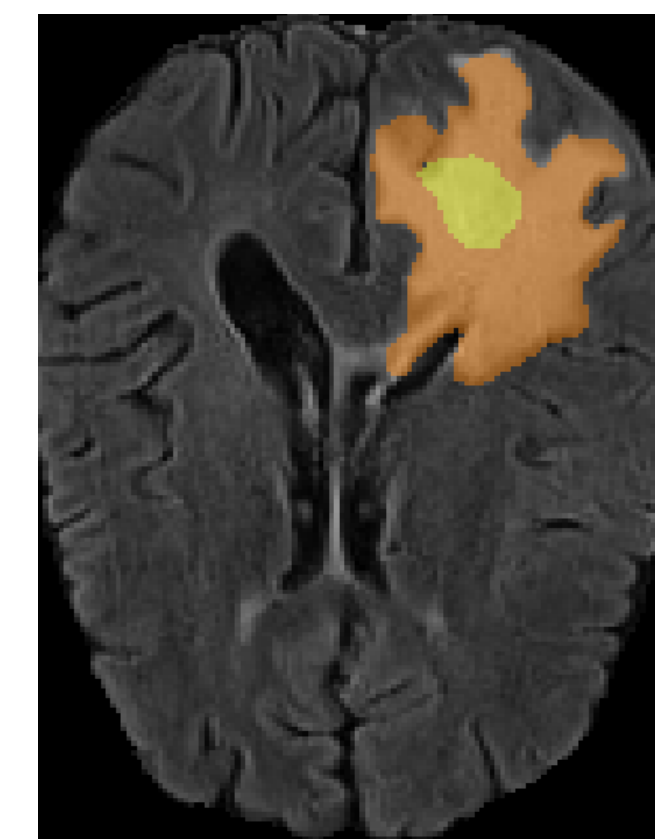
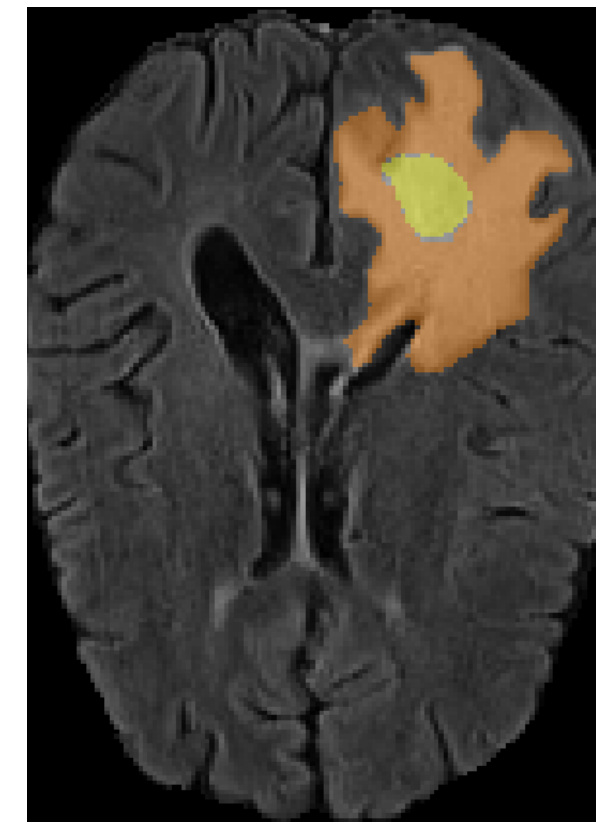
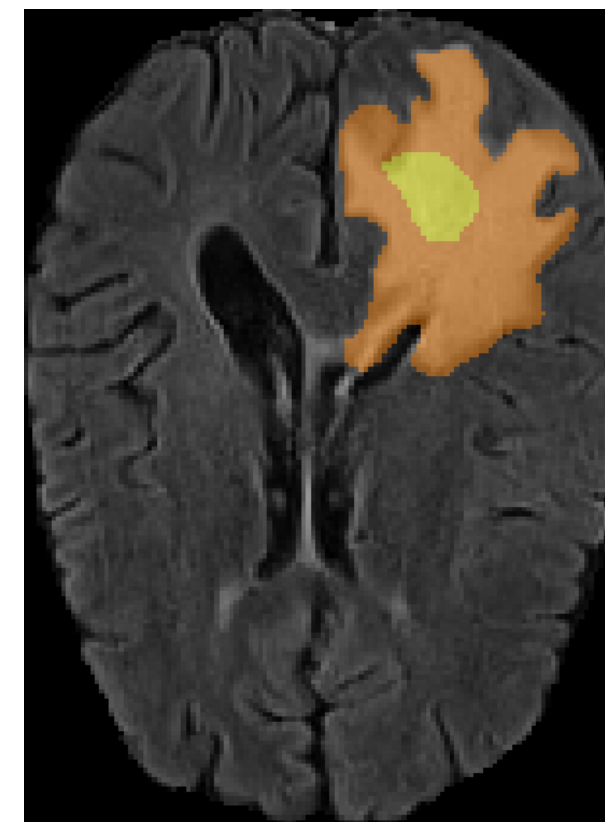
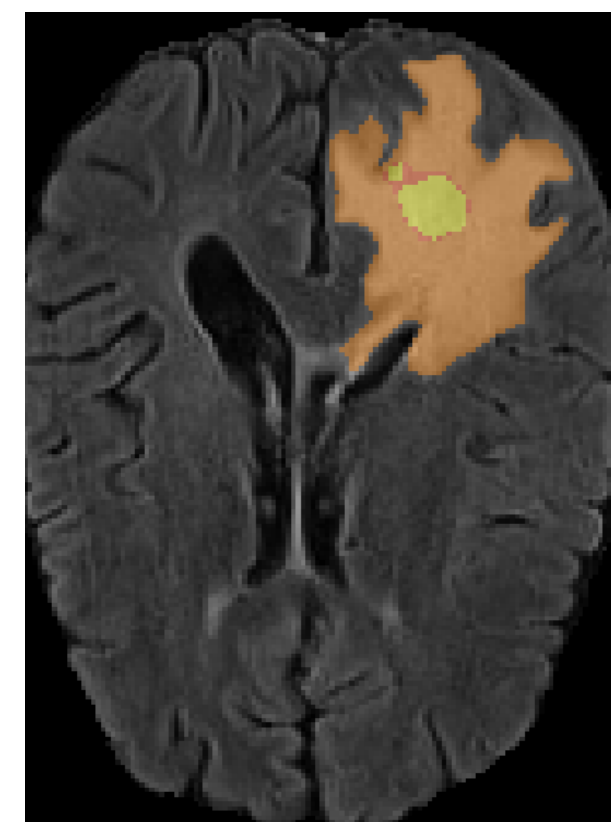
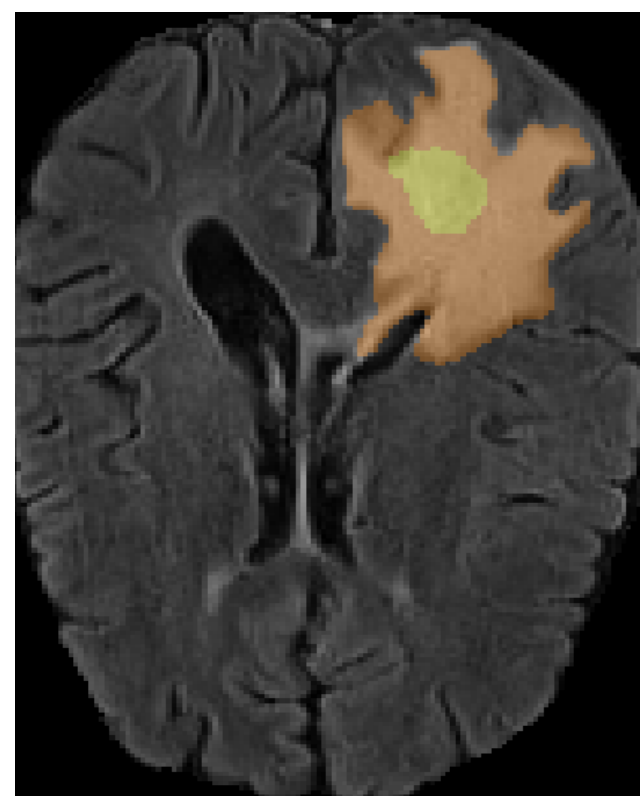
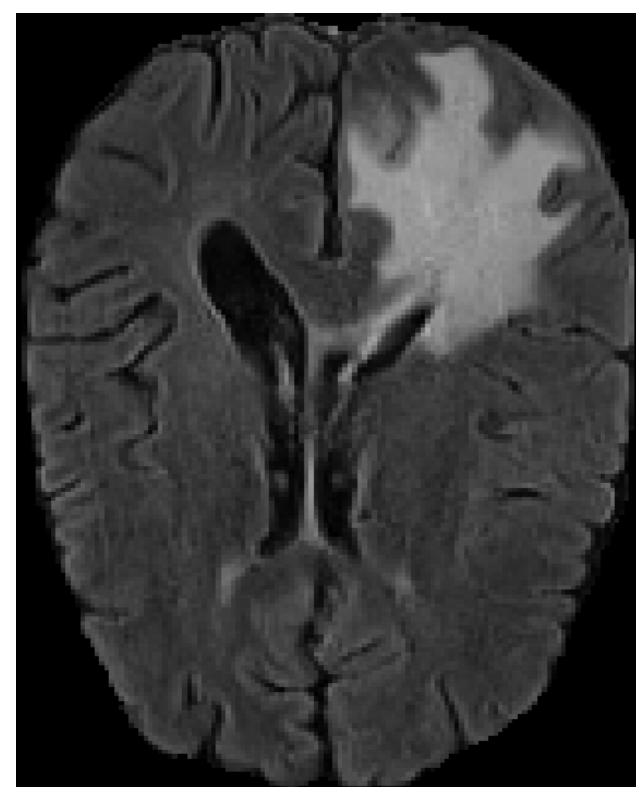
GBMNet18

Majority Voting

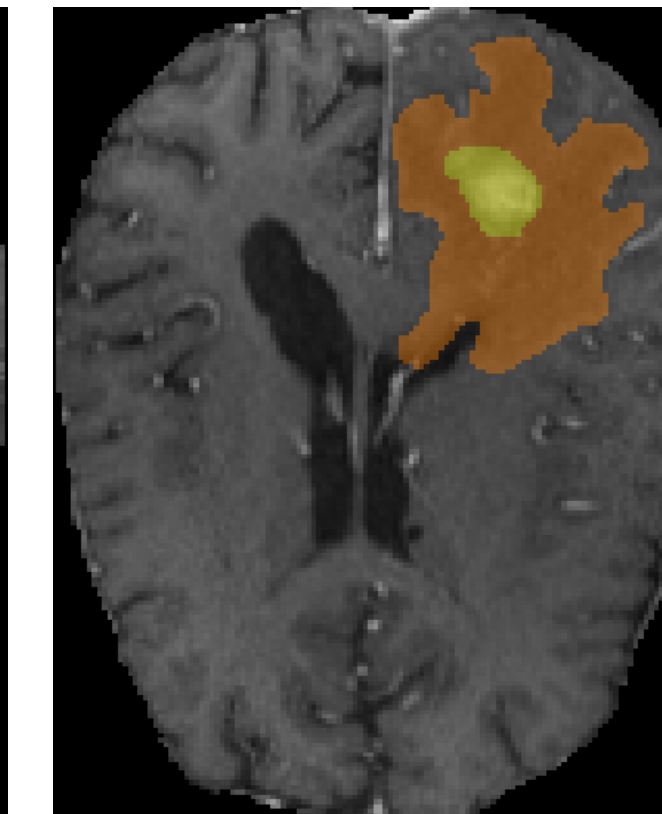
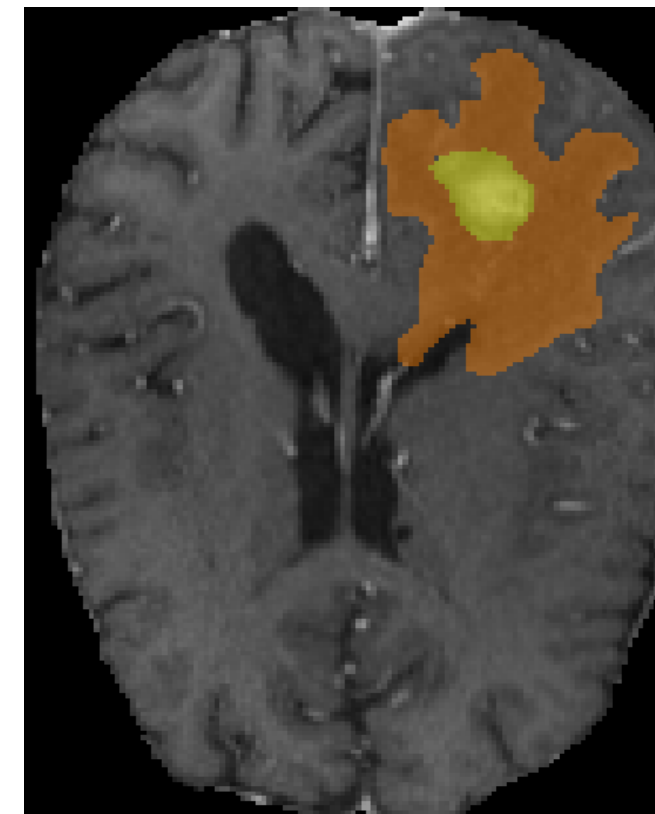
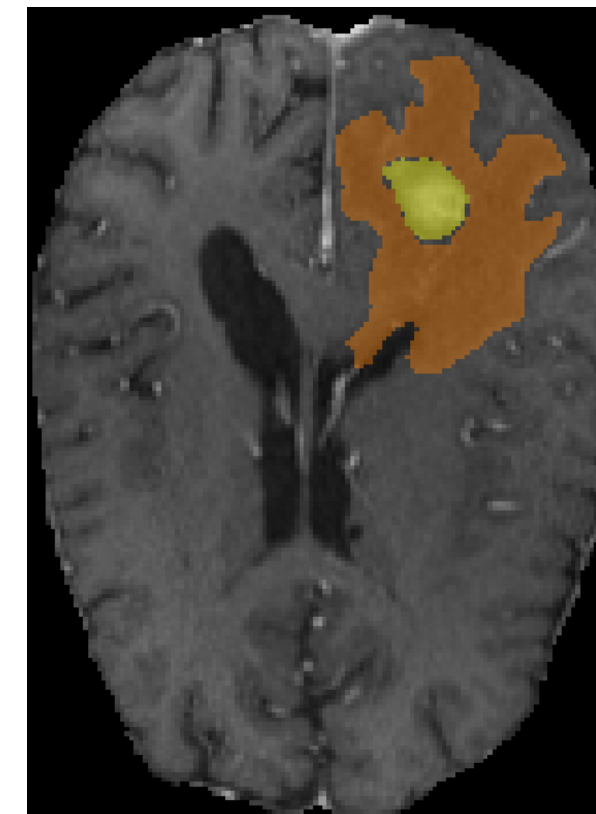
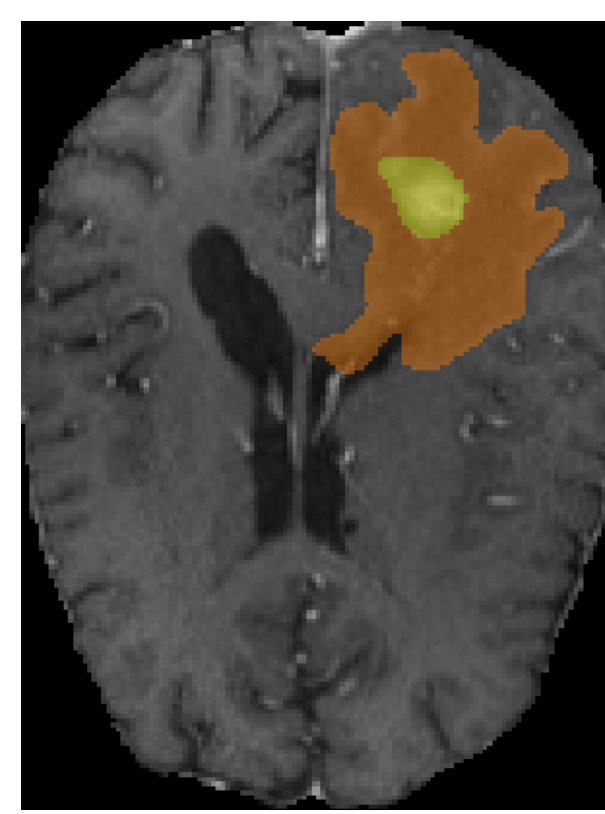
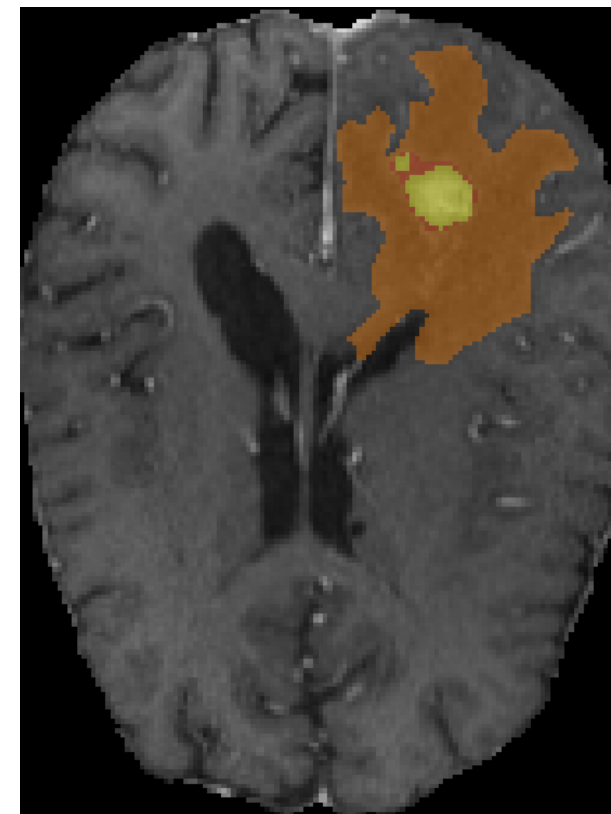
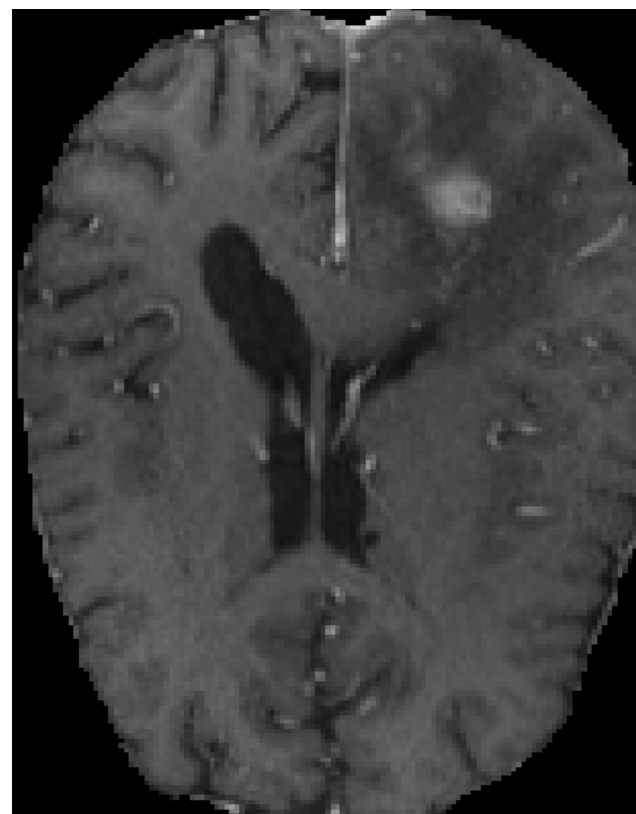
SIMPLE

STAPLE

Flair



T1ce



Inputs

DeepMedic

DKFZ

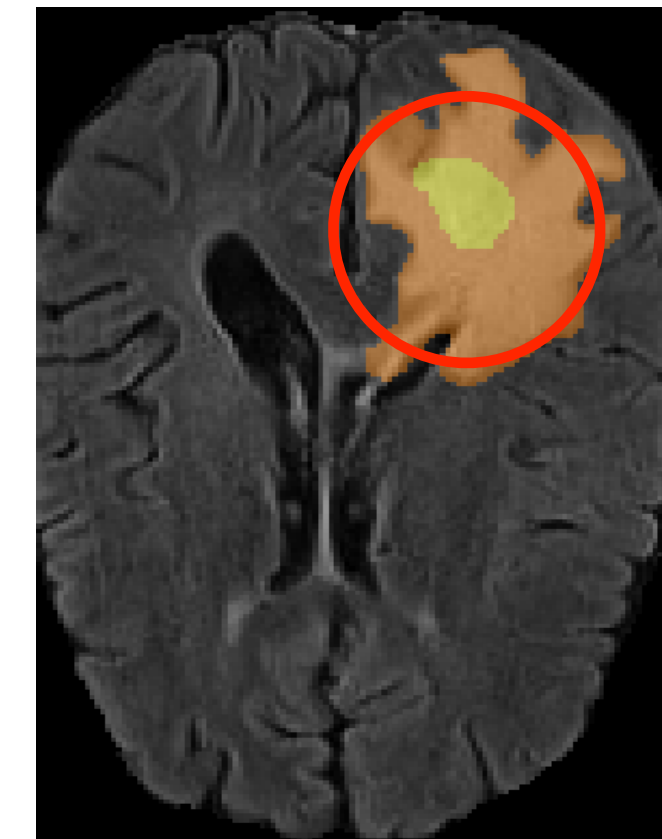
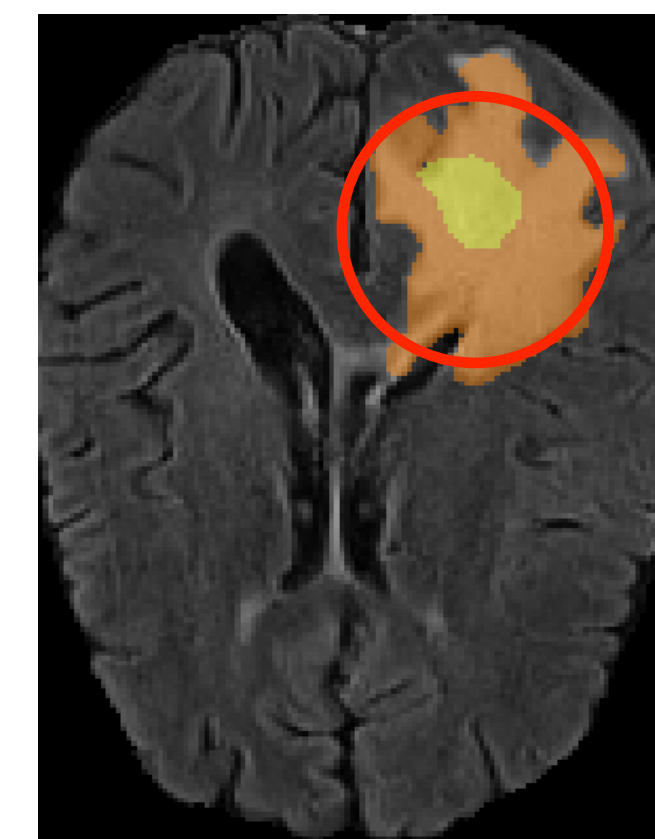
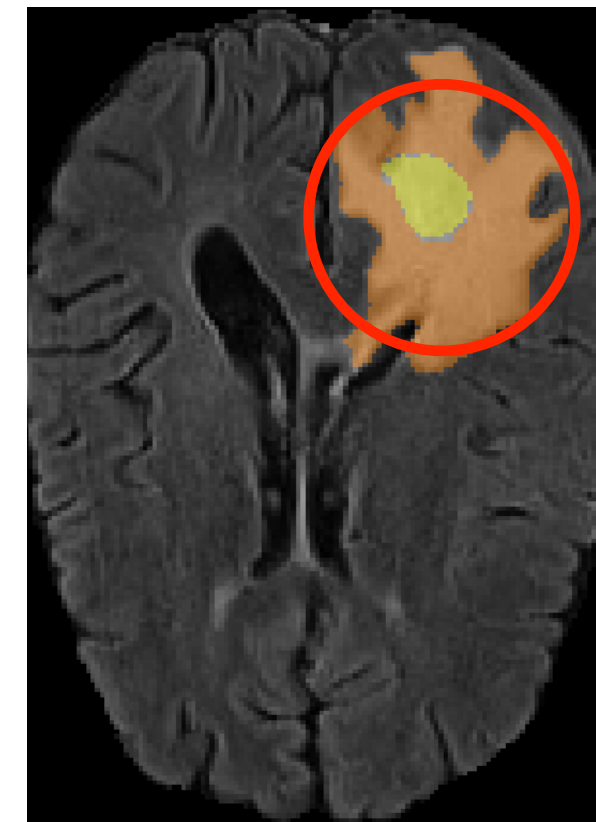
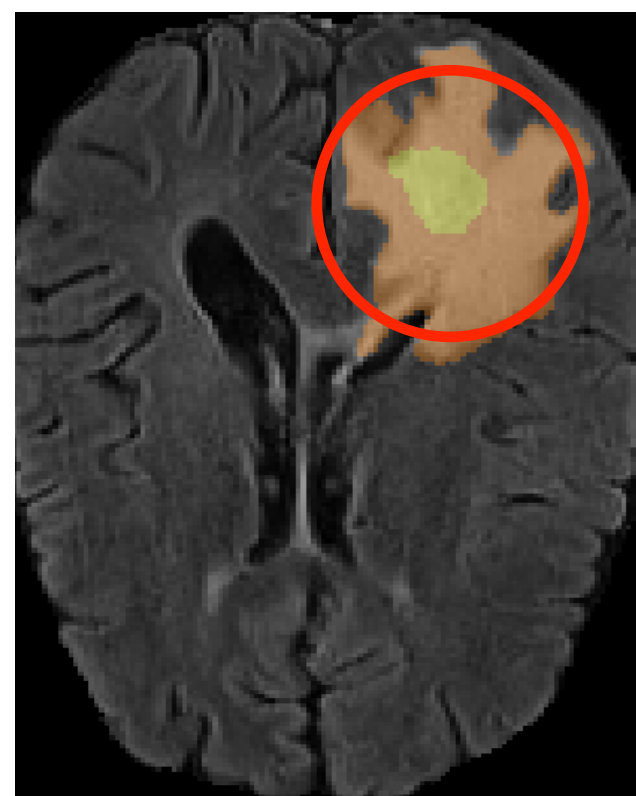
GBMNet18

Majority Voting

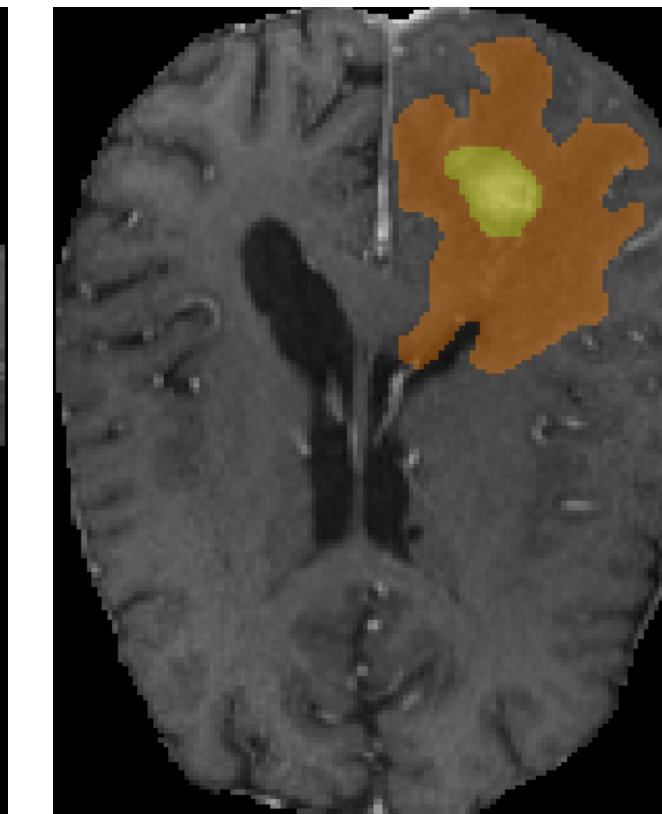
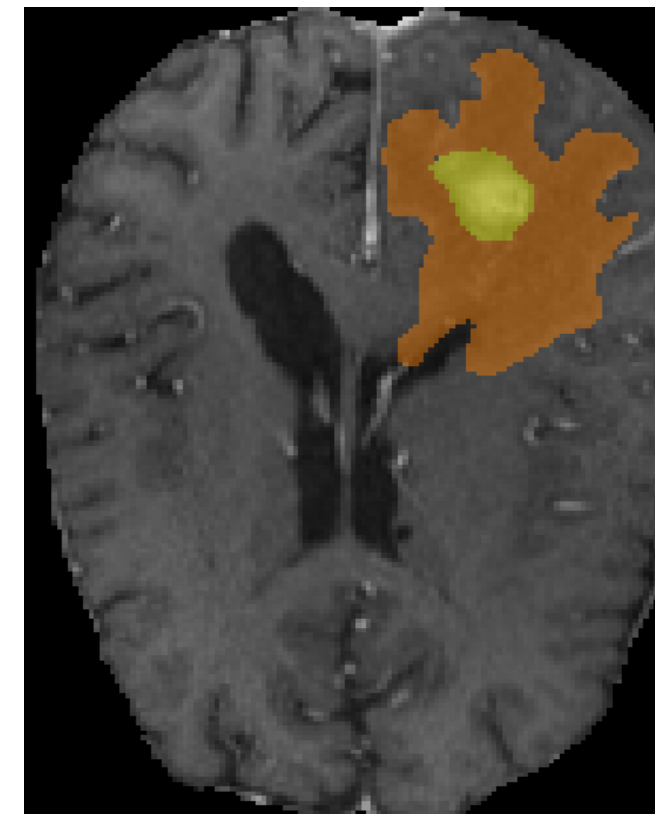
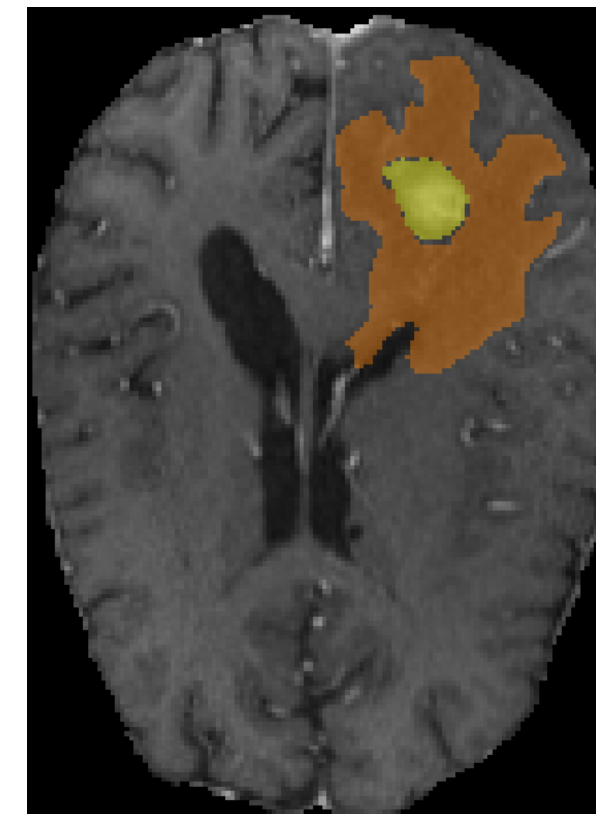
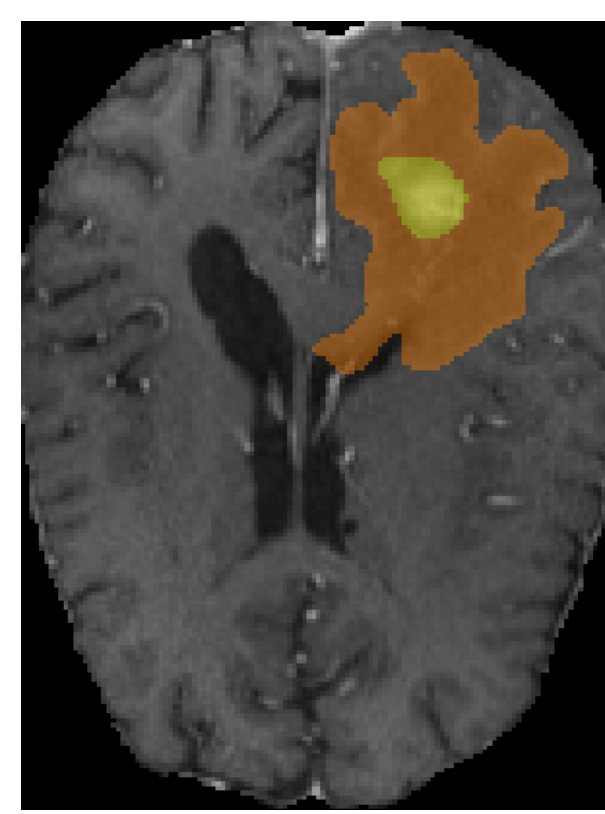
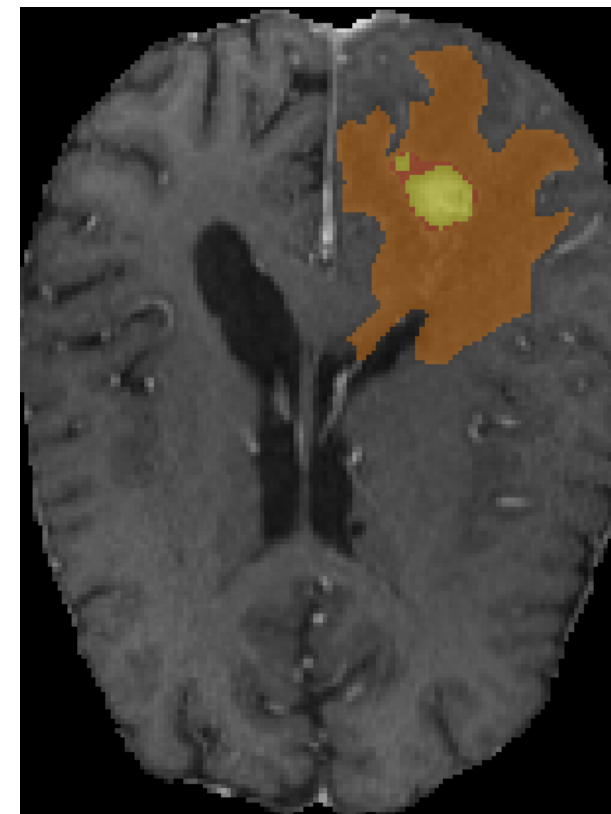
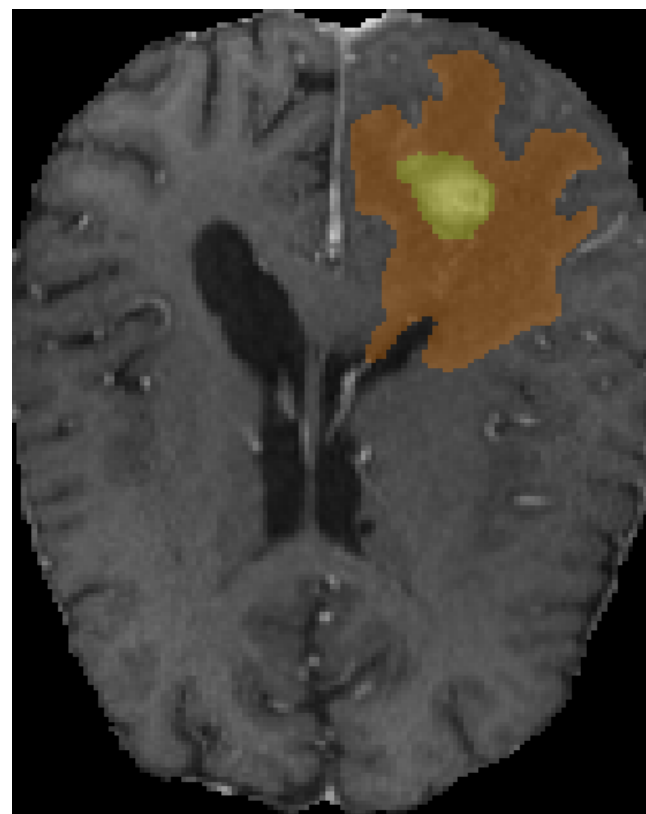
SIMPLE

STAPLE

Flair



T1ce



Inputs

DeepMedic

DKFZ

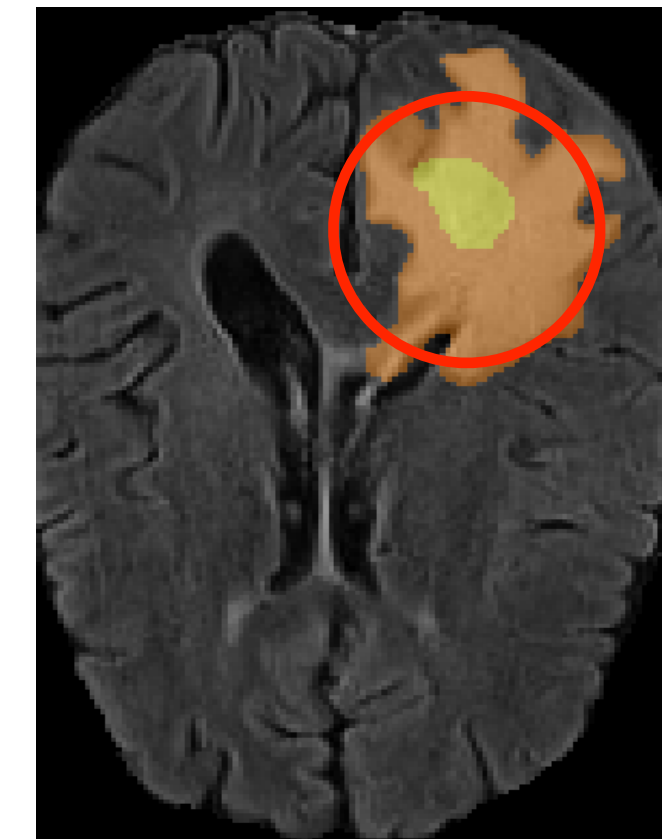
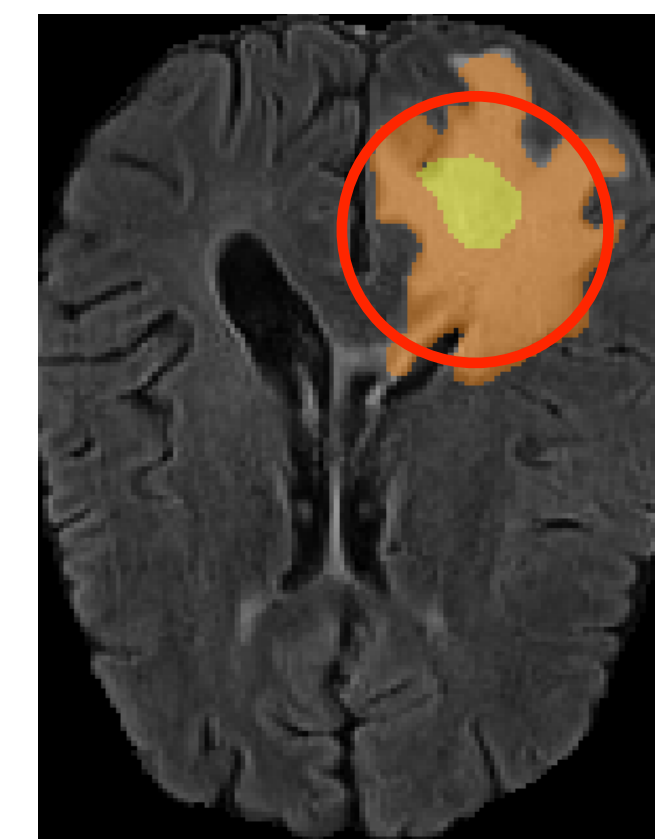
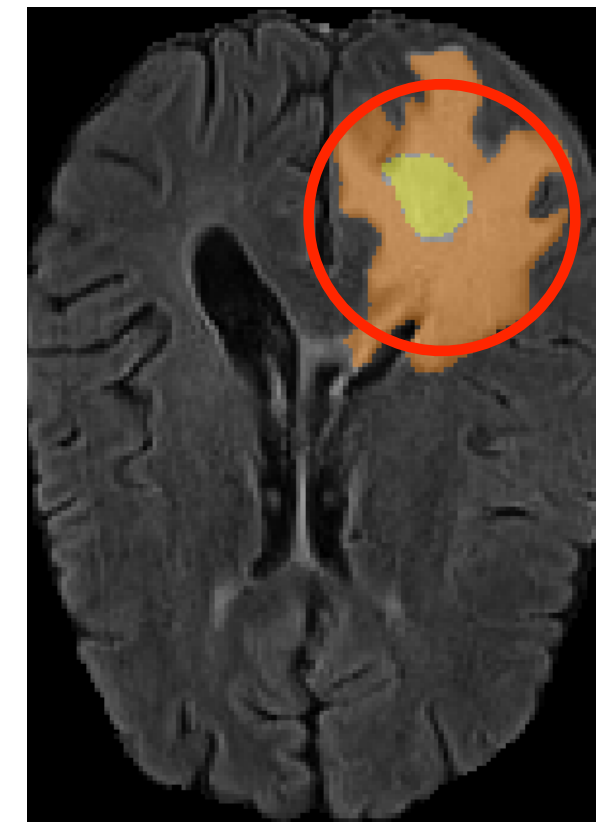
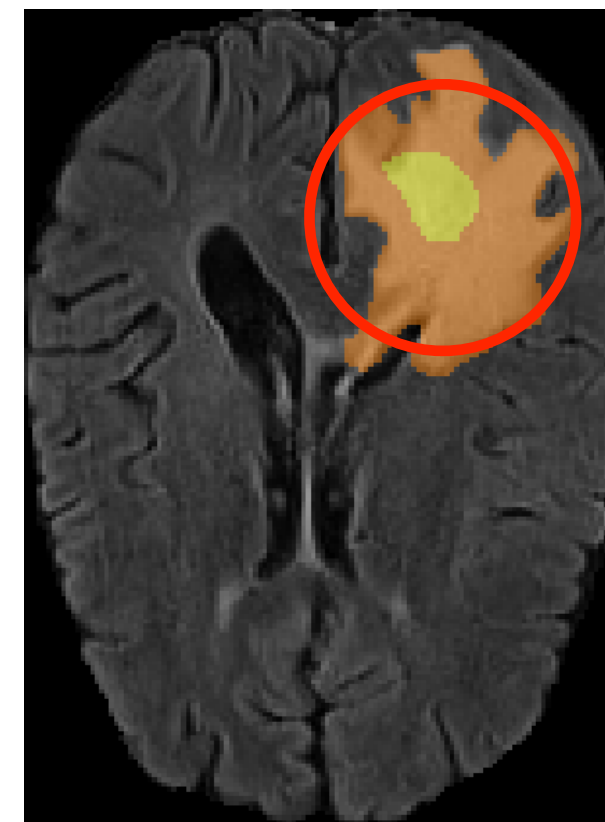
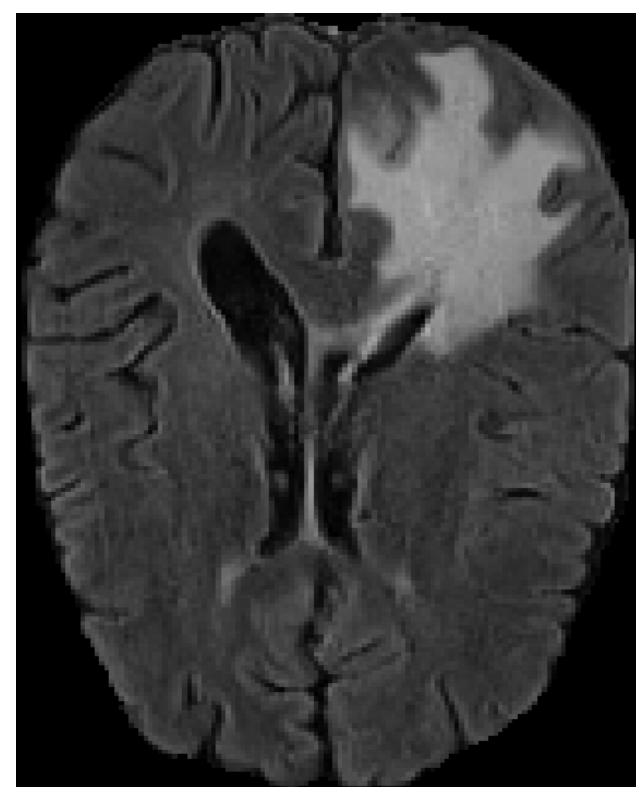
GBMNet18

Majority Voting

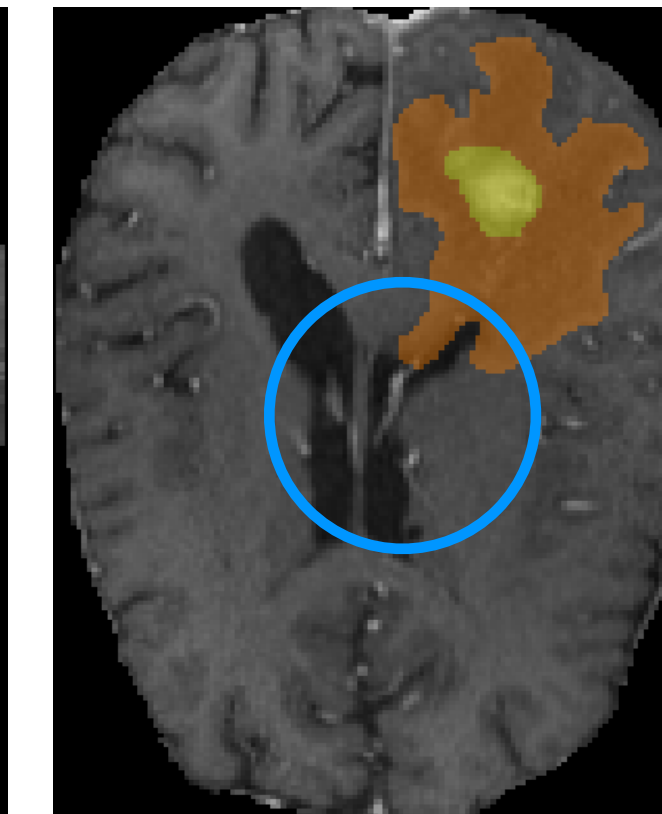
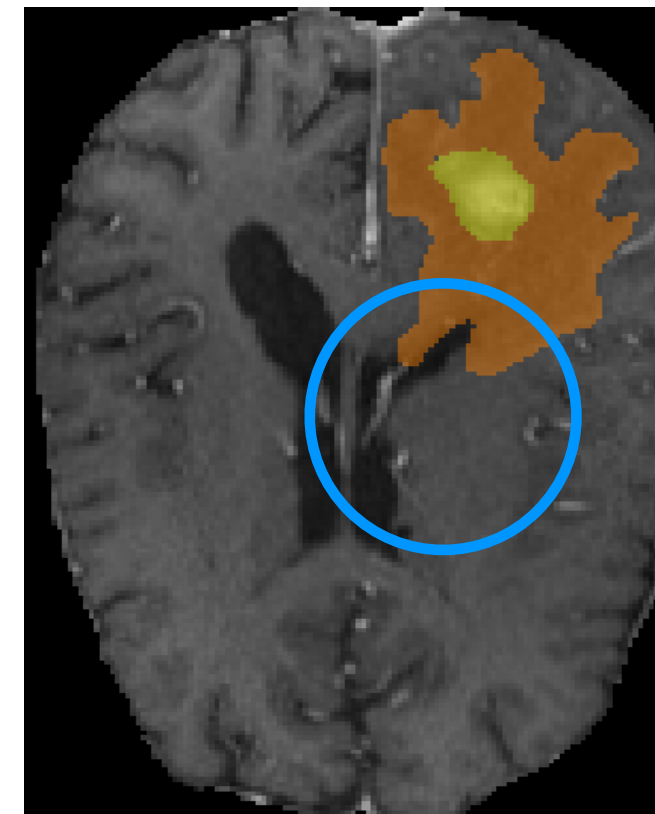
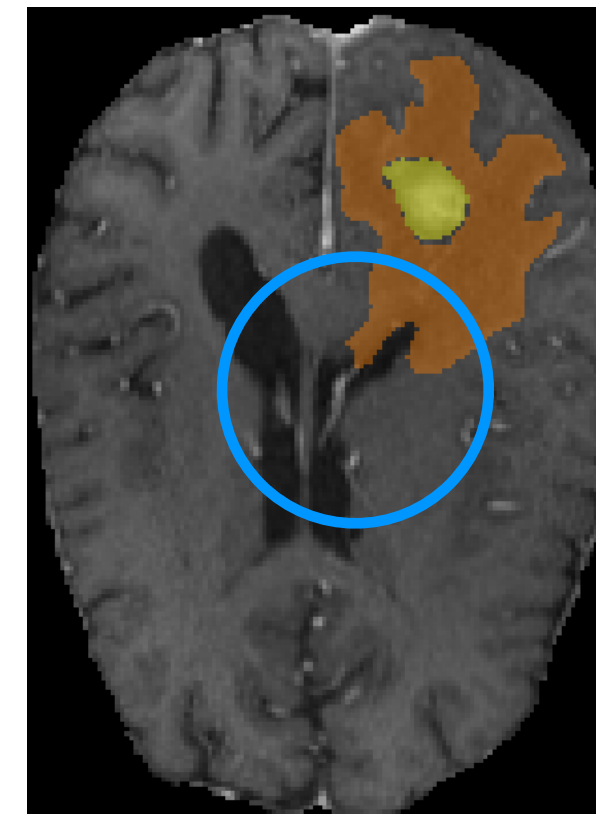
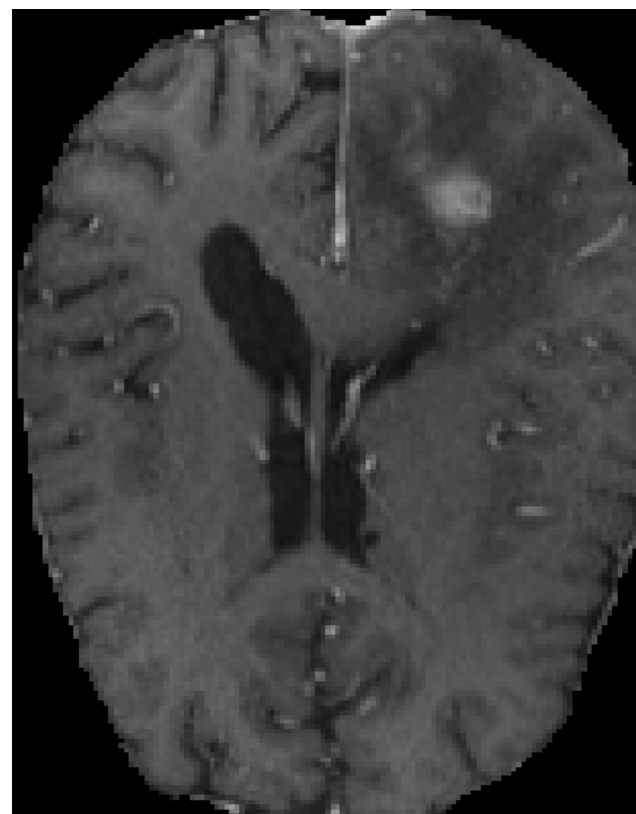
SIMPLE

STAPLE

Flair



T1ce



Thank you