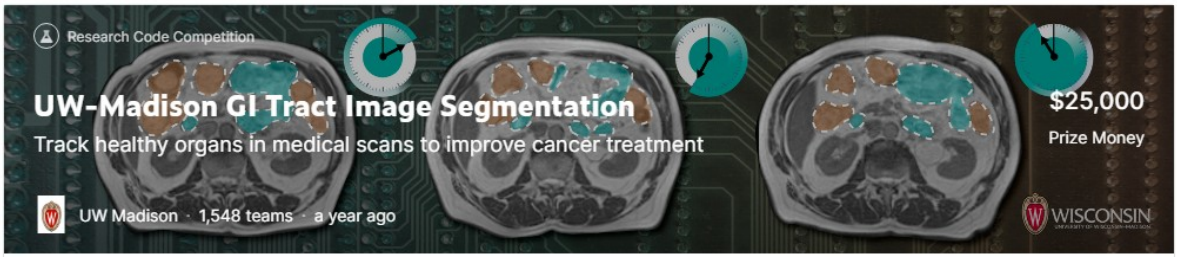
Image Segmentation - Current Techniques for Medical Applications



Chris Winsor
7/24/2023

This is a review of current architectures and techniques for image segmentation in medical applications. It includes:

- Review of current research (architectures and pre-trained models)
- Survey of submissions from the recent "UW-Madison GI Tract Image Segmentation" competition on Kaggle
- An implementation of UNet/EfficientNet segmentation model using the UW-Madison dataset.

The UW-Madison Kaggle competition ran from April 2021 through July 2022. The goal is to segment stomach and intestines from MRI scans. Reference https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/overview

Table of Contents

Segmentation Architectures (reverse chronological order)

The following is a review of selected segmentation architectures and related work. It is in reverse chronological order by date of publication.

Current research in image segmentation is currently split in two different tracks: Vision Transformers (ViTs) and ConvNets. The former applies the Transformer architecture from NLP and is considered newest and latest. ConvNets are considered older and more traditional but recent updates demonstrate performance on par, or better than, ViTs.

Yale Aneja Lab Study (Avesta et al., 2023)

- Avesta, A., Hossain, S., Lin, M., Aboian, M., Krumholz, H. M., & Aneja, S. (2023). Comparing 3D, 2.5D, and 2D Approaches to Brain Image Auto-Segmentation. *Bioengineering*, *10*(2), 181. https://doi.org/10.3390/bioengineering10020181

(Avesta et al., 2023) perform a pragmatic comparison of Capsule Network, UNet, nnUNet, when tasked with 2D, 2.5D, 3D segmentation in brain imaging. The following are evaluated:

- Dice score with full training data (3199 images)
- Dice score with limited training data (600, 240, 120, 60 images)
- Computational speed during training (time/example/epoch)
- Computational speed at runtime
- Computational memory (GPU MB) to train/deploy.

Dataset and Training Target:

- 3430 brain MR images (841 patients) from Alzheimer’s study[1]
- Model trained to segment ventricle, thalamus, hippocampus
- Preprocessing to voxels (643) or slice/s (N*642)

Compute:

- 16 GB GPU memory

Image Patch:

- Input 3D = 64x64x64 voxel. Input 2.5D = 64x64 (5 slices) Input 2D = 64x64.

Summary:

- The three architectures (Caps Net, UNet, nnUNet) performed similarly - within 1% of each other.
- 3D models gave highest Dice scores - around 4% to 5% higher than 2.5D. 2.5D had only marginally better performance than 2D. This persisted as training size decreased down to 60 images.
- 3D models converged faster and were faster during deployment.
- 3D models require up to 20 times more memory, however "Our results show that 3D CapsNets outperformed all 2.5D and 2D models while only requiring twice more computational memory than the 2.5D or 2D UNets or nnUNets. Conversely, 3D UNets and nnUNets required 20 times more computational memory compared to 2.5D or 2D UNets and nnUNets. "

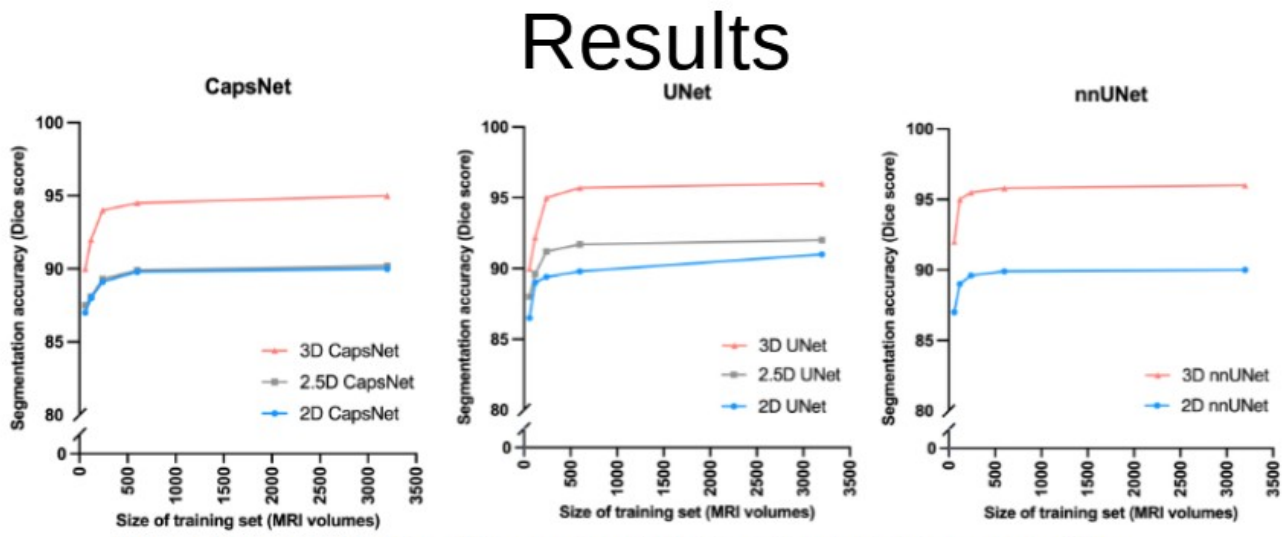


Figure 3. Comparing 3D, 2.5D, and 2D approaches when training data is limited. As we decreased the size of the training set from 3000 MRIs down to 60 MRIs, the CapsNet (**a**), UNet (**b**), and nnUNet (**c**) models maintained higher segmentation accuracy (measured by Dice scores).

Avesta, A., Hossain, S., Lin, M., Aboian, M., Krumholz, H. M., & Aneja, S. (2023). Comparing 3D, 2.5D, and 2D Approaches to Brain Image Auto-Segmentation. Bioengineering, 10(2), 181. https://doi.org/10.3390/bioengineering10020181

ConvNeXt (Facebook Research 2022)

- Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). *A ConvNet for the 2020s* (arXiv:2201.03545). arXiv. https://doi.org/10.48550/arXiv.2201.03545
- https://github.com/facebookresearch/ConvNeXt

ConvNeXt is a modernization of the ConvNet architecture. It introduces subtle but important advancements from vision transformers (ViTs) without wholesale adoption, and eliminates elements from the legacy ConvNet that are unnecessary. ConvNeXt perform on par with ViTs (e.g. Swin) on classification tasks, and beats ViTs on segmentation and localization tasks such as those common to medical imaging.

Facebook Research released a set of small and light pretrained ConvNeXt models intended as backbone for transfer learning.

Elements of design:

- GELU replaces ReLu
- Fewer activation blocks (single GELU per block)
- Single Batch Norm per block
- Layer Norm replaces Batch Norm
- Separate downsampling layers

Classification Performance:

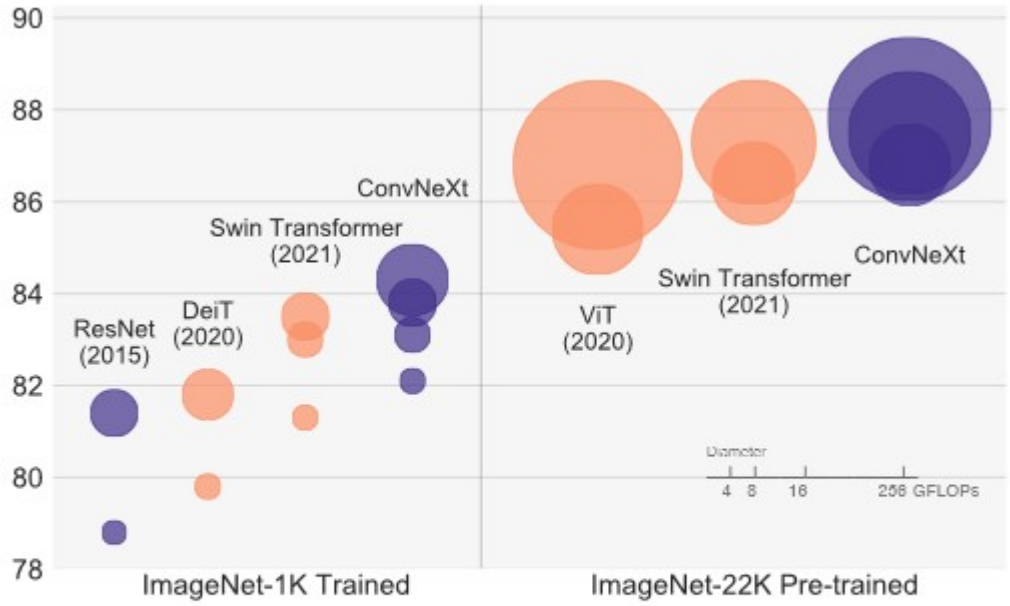


Figure 1. ImageNet-1K classification results for ● ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family.

Segmentation Performance:

backbone	FLOPs	FPS	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
Mask-RCNN 3× schedule								
○ Swin-T	267G	23.1	46.0	68.1	50.3	41.6	65.1	44.9
● ConvNeXt-T	262G	25.6	**46.2**	67.9	50.8	**41.7**	65.0	44.9
Cascade Mask-RCNN 3× schedule								
● ResNet-50	739G	16.2	46.3	64.3	50.5	40.1	61.7	43.4
● X101-32	819G	13.8	48.1	66.5	52.4	41.6	63.9	45.2
● X101-64	972G	12.6	48.3	66.4	52.3	41.7	64.0	45.1
○ Swin-T	745G	12.2	50.4	69.2	54.7	43.7	66.6	47.3
● ConvNeXt-T	741G	13.5	**50.4**	69.1	54.8	**43.7**	66.5	47.3
○ Swin-S	838G	11.4	51.9	70.7	56.3	45.0	68.2	48.8
● ConvNeXt-S	827G	12.0	**51.9**	70.8	56.5	**45.0**	68.4	49.1
○ Swin-B	982G	10.7	51.9	70.5	56.4	45.0	68.1	48.9
● ConvNeXt-B	964G	11.4	**52.7**	71.3	57.2	**45.6**	68.9	49.5
○ Swin-B‡	982G	10.7	53.0	71.8	57.5	45.8	69.4	49.7
● ConvNeXt-B‡	964G	11.5	**54.0**	73.1	58.8	**46.9**	70.6	51.3
○ Swin-L‡	1382G	9.2	53.9	72.4	58.8	46.7	70.1	50.8
● ConvNeXt-L‡	1354G	10.0	**54.8**	73.8	59.8	**47.6**	71.3	51.7
● ConvNeXt-XL‡	1898G	8.6	**55.2**	74.2	59.9	**47.7**	71.6	52.2

Table 3. **COCO object detection and segmentation results**

Swin Transformer (Microsoft Research Asia, 2021)

- Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* (arXiv:2103.14030). arXiv. https://doi.org/10.48550/arXiv.2103.14030

Swin (Shifted Windows) Transformer is a general purpose backbone for computer vision that adapts Transformer from language to vision. Swin features a hierarchical design and uses a shifted window scheme that limits self-attention to non-overlapping regions. As a result Swin achieves computational complexity linear with image size.

Query patches within a window share the same key set (query, key, value from Transformer) leading to efficiency.

Split RGB into non-overlapping patches (see ViT) such as 4x4x3 (where "3" is RGB).
Patch treated as "token"
Stage 1: multiple Swin Transformer Blocks applied maintaining (H/4, W/4)
Patch merging layer reduces token (2x, 2x) with output dimension = 2C
Stage 2: Swin Transformer Blocks applied (H/8, W/8)
Stage 3 and 4 are /16 and /32

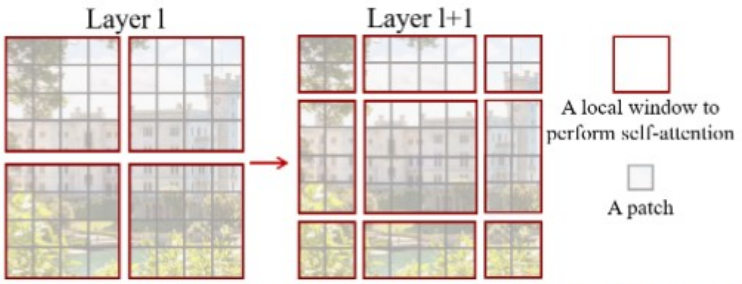


Figure 2. An illustration of the *shifted window* approach for computing self-attention in the proposed Swin Transformer architecture. In layer l (left), a regular window partitioning scheme is adopted, and self-attention is computed within each window. In the next layer $l + 1$ (right), the window partitioning is shifted, resulting in new windows. The self-attention computation in the new windows crosses the boundaries of the previous windows in layer l, providing connections among them.

CapsNet (CRCV 2021) and 3D-CapsNet (Yale 2022)

- Avesta, A., Hui, Y., Aboian, M., Duncan, J., Krumholz, H. M., & Aneja, S. (2022). 3D Capsule Networks for Brain Image Segmentation (p. 2022.01.18.22269482). medRxiv. https://doi.org/10.1101/2022.01.18.22269482
- LaLonde, R., Xu, Z., Irmakci, I., Jain, S., & Bagci, U. (2021). Capsules for biomedical image segmentation. Medical Image Analysis, 68, 101889. https://doi.org/10.1016/j.media.2020.101889

3D Capsule Networks (2022) are an extension to 2D CapsNets (2021).

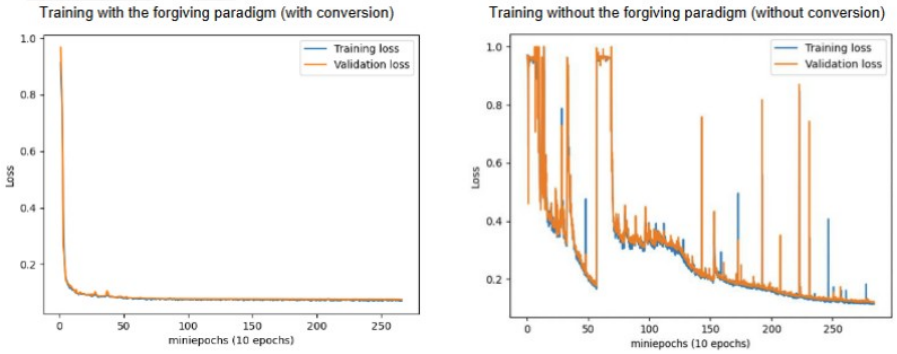
Key novel design elements:

- Capsule encodes Structure + Pose (size, curvature, location, etc)
- Transform converts pose of parts to pose of whole
- Detection of whole is based on parts/transform
- Many design choices (network architecture, hyperparameters, loss functions - see Appendix 4 in paper)
- Forgiving Paradigm: final layer activation stabilizes training

Results:

- Segmented ventricle, thalamus, and hippocampus with Dice scores within 1% of of UNets and nnUNets
- Significantly outperformed UNets in images that are not well-represented in the training (30% higher Dice scores).
- **Memory required less than a tenth of that for UNets or nnUNets**
- More than 25% faster to train compared with UNet and nnUNet.

3D CapsNet (Forgiving Paradigm)



Code and Notes:

- Code used to train and test our models, our pre-trained models, and a sample MRI are available on our lab's GitHub page: https://github.com/Aneja-Lab-Yale/Aneja-Lab-Public-CapsNet
- Note Appendix 4 (Design Choices to Develop…) in paper referenced below as it discusses options/alternatives explored.
- Also see the related work on the github... https://github.com/Aneja-Lab-Yale/Aneja-Lab-Public-3D2D-Segmentation

EfficientNet (Google Research, 2020)

- Tan, M., & Le, Q. V. (2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks* (arXiv:1905.11946). arXiv. https://doi.org/10.48550/arXiv.1905.11946
- Xie, Q., Luong, M.-T., Hovy, E., & Le, Q. V. (2020). Self-Training With Noisy Student Improves ImageNet Classification. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10684–10695. https://doi.org/10.1109/CVPR42600.2020.01070

EfficientNet is:

- A "compound scaling method" to increase CNN performance by adjusting depth/width/resolution in a prescribed manner. Given a desired increase in performance the algorithm recommends parameters. The algorithm can be applied to ResNet and MobileNet.
- A set of pre-sized networks that have been performance optimized (performance / parameter count).

EfficientNets have been shown to achieve same or better performance with an order of magnitude fewer parameters.

EfficientNet B0 through B7 are in the original paper. EfficientNet-L2 was introduced by Xie et al (2020) and is "wider and deeper than EfficientNet-B7 but uses a lower resolution, which gives it more parameters to fit a large number of unlabeled images. Due to the large model size, the training time of EfficientNet-L2 is approximately five times the training time of EfficientNet-B7."

UNet++ (2018 Arizona State University)

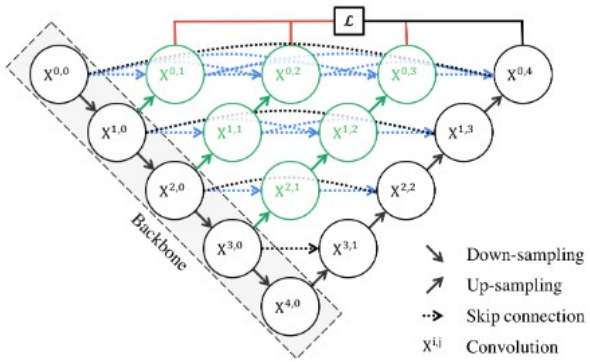
- Zhou, Z., Siddiquee, M. M. R., Tajbakhsh, N., & Liang, J. (2018). UNet++: A Nested U-Net Architecture for Medical Image Segmentation (arXiv:1807.10165). arXiv. https://doi.org/10.48550/arXiv.1807.10165

UNET++ is an encoder/decoder network for image segmentation. As compared to earlier work UNet++ redesigns the skip pathway. The premise is that skip connections have been shown as key for detail and multi-instance separation, so by enhancing the skip connections UNet++ aims to "reduce the semantic gap between the feature maps of the encoder and decoder sub-networks".

The image (from the paper) shows (BLACK) U-Net backbone and (GREEN/BLUE) skip connections introduced by UNet++.

UNet++ is shown to out-perform U-Net and wide U-Net by 3.9 and 3.4 points respectively. The application is medical segmentation from CT scan and video. The metric is IoU.

Code (torch, keras) are available at https://github.com/MrGiovanni/UNetPlusPlus



nnU-Net (German Cancer Research Center DKFZ, 2018)

- Isensee, F., Petersen, J., Klein, A., Zimmerer, D., Jaeger, P. F., Kohl, S., Wasserthal, J., Koehler, G., Norajitra, T., Wirkert, S., & Maier-Hein, K. H. (2018). *nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation* (arXiv:1809.10486). arXiv. https://doi.org/10.48550/arXiv.1809.10486

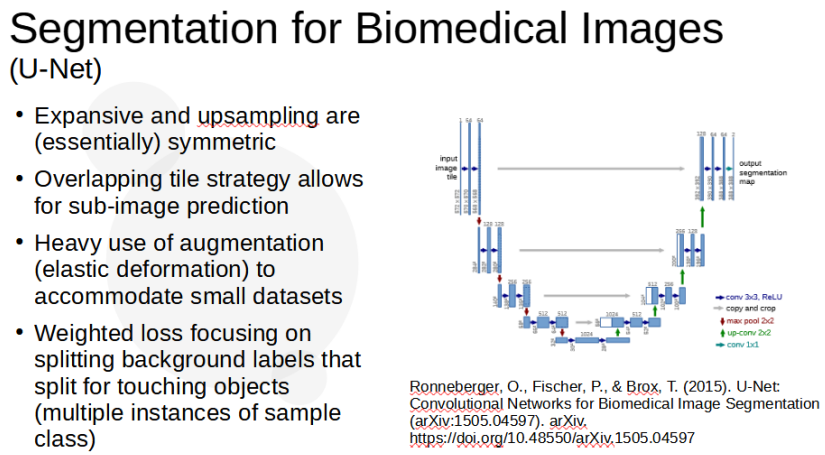
"The present paper introduces the nnU-Net ('no-new-Net'), which refers to a robust and self-adapting framework on the basis of 2D and 3D vanilla U-Nets. We argue the strong case for taking away superfluous bells and whistles of many proposed network designs and instead focus on the remaining aspects that make out the performance and generalizability of a method."

"We evaluate the nnU-Net in the context of the Medical Segmentation Decathlon challenge, which measures segmentation performance in ten disciplines comprising distinct entities, image modalities, image geometries and dataset sizes, with no manual adjustments between datasets allowed." (reviewer's underlining)

U-Net (University of Freiburg, 2015)

- Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation* (arXiv:1505.04597). arXiv. https://doi.org/10.48550/arXiv.1505.04597
- https://en.wikipedia.org/wiki/U-Net

U-Net is a convolutional network for biomedical image segmentation. It consists of contracting and expansive paths. The expansive path uses upsampling to increase resolution of the output, i.e, to make a class prediction for each pixel.



Kaggle Submissions Detailed Reviews

Below are reviews of the the "Nth Place Solution" in the "Discussion" tab from the UW-Madison kaggle competition.

1st Place Solution (CARNO ZHAO)

https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/discussion/337197

OVERVIEW:
Three groups of models:

- CLS (classification) group classifies whether the prediction of a slice should be empty or not.
- SEG (segmentation) group and 3D group do segmentation for slices that should be predicted.

Code available: Various links - see https://www.kaggle.com/code/carnozhao/1st-place-winning-solution?scriptVersionId=100822691

CLS Group:

- trained w/ whole dataset (not folds)
- Model = 4 unets with efficientnet (4,5,6,7)
- Training = Public mmsegmentation pipeline (see above)
- If the CLS is deemed negative no further processing is done (generate zero masks)

SEG Group:

- EfficientNet/Unet *with annotated slides* (???)
- 5 Unets (4,5,6,7)
- 2 UPerNet (see https://huggingface.co/docs/transformers/model_doc/upernet)

3D Group:

- Public MONAI pipeline (preprocessing) see https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/discussion/325646

Additional Info (in acknowledgments section):

- And also thanks to other competitors that inovated us by their previous solutions: [1], [2], [3], from which our classification-segmentation two-stage pipeline and training tricks comes.
- And @yiheng, without his excellent 3D monai training pipeline, we can't get such a high score.

2nd Place Solution ("ISHIKEI")

2-stage pipeline:

- Stage 1: predict (+/-) slices
- Stage 2: segmentation on slices predicted as (+)

Ensemble of 2.5D and 3D
2.5D Model
5 slice, 512x512 pixels
Crop by YOLOv5
The structure is "backbone,
Stage 1 backbone = Efficientnet B4, Swin Base
Stage 2 backbone = Efficientnet L2, ConvNetXt SL, Swin Large
Decoder = UPerNet

3rd Place Solution ("HE")

https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/discussion/337468
OVERVIEW:

- Similar to 2nd place solution but without 3-D models.
- Detector to detect main area
- Classifier to determine if segmentation necessary
- Segment image (if positive)

Code available: I am cleaning my code recently, you can refer to our previous solution at this, the training pipline is similar. https://www.kaggle.com/competitions/siim-acr-pneumothorax-segmentation/discussion/108009 (SIIM-ACR Pneumothorax Segmentation)

8th place solution (EVGENII KONONENKO)

https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/discussion/337359

Code available: No

Summary:

- An ensemble of 6 models: three 2.5D, three 3D

- Folding: standard 5 folds grouped by case, with different splitting for each ensemble model
- Segmentation Loss: Focal + Dice
- Validation metric: Dice
- Optimization: AdamW
- Augmentations: albumentations (flip, elastic/grid distort, blue/noise, brightness/contrast)

Training strategy:

- classification (predict whether a slide will have annotation)
- segmentation (predict masks)

All models Unet variations, so simply add classification at head of encoder Unet to classify presence of annotation.

Ambiguous annotation (absence of annotation) near first and last annotated slices - simply ignore such wrong slices when computing segmentation loss (for classification loss they are still used)

2.5D models:

all are Unet++ from smp (segmentation models PyTorch)

5 folds for each model, no TTA

- Unet++ with efficientnet-b8 with classificaiton head for 1 output
- Unet++ with efficientnet-b4 with classification head for 3 outputs
- Unet++ with hrnet-w44 with classification head for 1 output a

3D Models:

all 3D models are ResidualUNet3D from pytorch-3dunet

5 folds for each model, no TTA

- ResidualUNet3D with f_maps
- ResidualUNet3D with f_maps
- ResidualUNet3D with f_maps

Ensemble:

- ensemble 6 models with almost equal weights. Three classification outputs predict annotation presence)

5th Place Solution (Qishen Ha)

Code available: No

key contribution is lifting ensemble performance by diversity

- 2.5D models (9) based on:
  - https://github.com/qubvel/segmentation_models.pytorch (smp)
  - https://github.com/rwightman/pytorch-image-models
  - https://github.com/sithu31296/semantic-segmentation
- 3D models (3) models based on:
  - https://github.com/qubvel/segmentation_models.pytorch
  - https://github.com/rwightman/pytorch-image-models
  - https://github.com/Project-MONAI/MONAI

11th Place Solution (NGUYENTHANHNHAN)

Code available: https://www.kaggle.com/code/andy2709/fork-of-best-public-kernel-ef10e0/notebook

https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/discussion/337193

2.5D models (7)

15th Place Solution ("YU4U") incl. "key ideas"

https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/discussion/337189

Code available: No

Summary:

- Preprocess - crop low intensity area
- Modeling
  - 2.5D models
  - 3D models
  - --> Label truncation prediction models (separate model)
- Postprocess
  - remove predicted masks using truncation prediction
  - remove top/bottom two pixels

Implementation (UNet/EfficientNet)

We reproduce a UNet/EfficientNet implementation using GI-Tract Segmentation data and published code.

Project Summary:

- Total development time: 6 weeks at about 15 hours/week (July 10 through August 21 at 2 days/week).
- Platform = NVidea GeForce RTX 3060 (12GB), Ubuntu 20.04 on WSL/Windows11. 64G memory, OS on SSD, data on platter.
- DataSet (preprocessing): Classes/structure provided by Torch. This includes DataSet, Dataloader, mm-segmentation and many details.
- Reference: from ZHAO at https://github.com/CarnoZhao/Kaggle-UWMGIT/blob/kaggle_tractseg/data_preprocess.py

Dataset:

- Image: 310x360, 16 bit grayscale.
- Annotation: RLE-encoded mask
- Train data is 85 cases where a case is a person. 38,496 images, each with 3 segmentation classes (stomach, small_bowel, large_bowl). 4.7GB (total)
- Test data is entirely unseen, about 50 cases.
- Each case includes a handful of scan sets - in the train dataset this is between 1 and 6 - each is associated with a day on which the scan set was taken. A scan set is typically 432 images however some scan sets violate this and have an odd number of scans. Specifically 7 of the training cases have a number of scans that is not an even multiple of 432.
- Because each case shows up on several days care must be taken to avoid data leakage during validation. This is done by using scikit-learn GroupKFold as described below.

Preprocessing Details

Here are the detailed steps and code review.

Download of Dataset

The UW-Madison GI-Tract data is downloaded and unzipped. Folders are organized by "case" (person) and within that by "day" (day on which scan was taken), and within that "scans" (the 144 .png files that constitute the scan).

The dataset provides a file (train.csv) which identifies id (folder/file), class (type of segmentation), and segmentation for some of the images (RLE encoding). An example is below showing two slices of data for a single case where segmentation is provided for the second slice.

```
case123_day20_slice_0104,stomach,
```

```
case123_day20_slice_0105,large_bowel,12377 7 12637 15 12869 4 12880 4 12900 19
13132 10 13144 8 13163 23 13396 23 13426 27 13661 60 13735 4 13926 80 14191 82
14456 84 14722 84 14987 85 15253 85 15518 85 15784 85 16049 86 16314 86 16579 87 1
6844 87 17109 55 17173 23 17374 45 17444 15 17640 32 17713 8 17906 25 17982 1 18171
23 18437 22 18703 22 18970 20 19236 20 19502 20 19769 18 20035 18 20302 17 20568 17
20834 17 21100 18 21366 18 21632 18 21898 17 22164 17 22430 16 22696 15
22961 15 23227 15 23492 15 23758 15 24024 15 24158 5 24290 15 24423 7 24556 16
24688 8 24822 16 24954 9 25087 17 25219 1 0 25353 17 25486 10 25618 17 25752 10
25884 15 26018 11 26150 15 26284 15 26415 15 26551 15 26682 14 26817 15 26948 13 2
7083 15 27215 12 27349 15 27482 11 27615 15 27749 9 27881 15 28017 4 28147 15 28413
14 28679 14 28945 13 29210 13 29476 13 29741 13 30007 12 30272 13 30537 12 30802 12
31067 13 31332 13 31598 12 31863 13 32129 12 32395 12 32661 11 32927 10 33193 10
33458 10 33723 10 33989 10 34255 9 34522 8 34788 6 35057 1
```

data_preprocessing.py

The following is line-by-line review of the data_preprocessing.py script.

Conda environment is "pytorch_o_cv2_u_madison_mmcv_via_mim"

Lines 29-42

The train.csv is read into a DataFrame "df_train".

"patient" and "days" columns are established from "id" and summary info is written to output

33,913 have segmentation data, 81,575 do not have segmentation data, total is 115,488 - each image slice is presented as 3 rows: segmentation for large_bowel, small_bowel, stomach. In other words there are 38,496 slices (115,488/3).

Lines 44-55

Image files (.png) from the "train" folder are listed. There are exactly 38,496 (satisfying). The file name identifies the image dimensions (X/Y) as well as pixel spacing which takes the format "1.50". To the dataframe is added columns "slice, size X/Y, spacing X/Y, image_files. Each is duplicated 3 times (np.repeat) along axis=0 meaning 3 rows are created for each to match above.

Lines 57-87

Folders "mmseg_train/images and "mmseg_train/labels" are created.

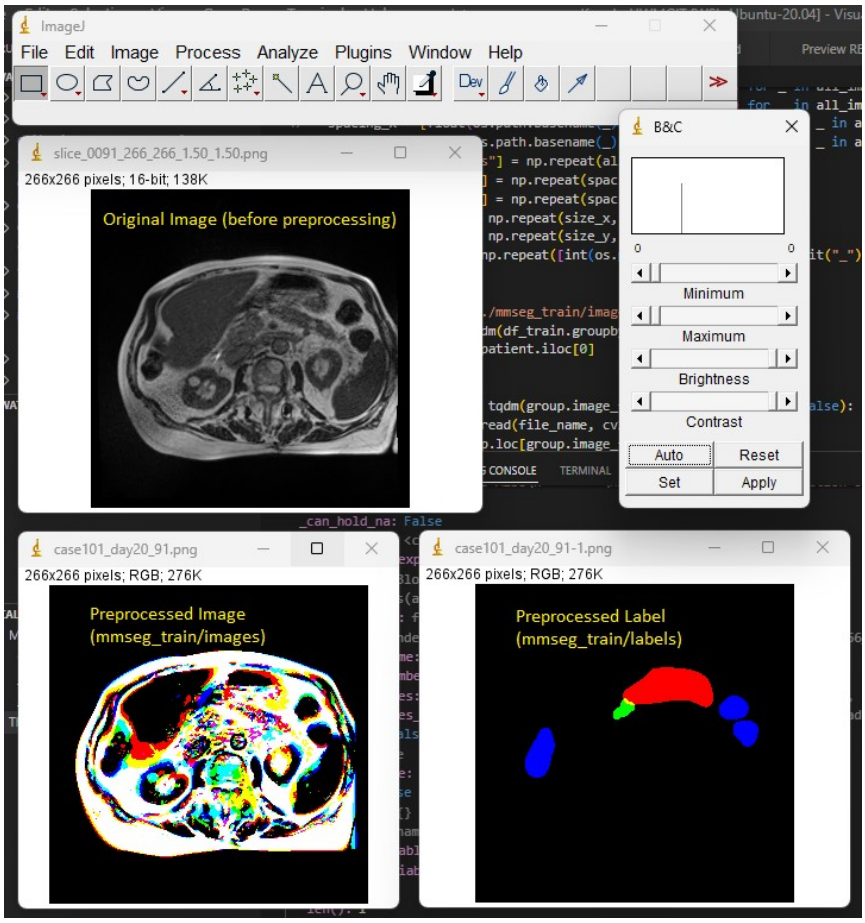
The dataframe above is grouped by "days" (where a "day" is a patient-day). Thus each group is all the images for one patient-day.

(lines 63-74) For each image in the patient-day is performed cv2.imread() giving a 266x266 numpy array of uint16. Then the segmentation data for the image is considered, if it is NaN then the mask is set "0", otherwise each run-length-encoded segment is decoded and mask set to "1". The images and masks for each patient are stacked - so for example this patient (case 101) had 144 images:

```
imgs.shape
(144, 266, 266)
msks.shape
(144, 266, 266, 3)
```

Some transpose is done (unexplained in lines 79-83) and images/labeles are written to file (cv2.imwrite) to folders "mmseg_train/images" and "mmseg_train_labels".

Thre are 38,496 images and 38,496 labels with example below.



(Exercise)

We wish to identify images that contain segmentation for all 3 types (stomach, upper, lower bowels).

```
>> criterion = df_train.loc[:, 'segmentation'].isna() == 0
>> df2 = df_train[criterion].groupby('image_files')['id'].count()
>> type(df2)
<class 'pandas.core.series.Series'>

>> y = df2==3

>> df2[y]
image_files
./train/case101/case101_day20/scans/slice_0091_266_266_1.50_1.50.png    3
./train/case101/case101_day20/scans/slice_0092_266_266_1.50_1.50.png    3
./train/case101/case101_day20/scans/slice_0093_266_266_1.50_1.50.png    3
./train/case101/case101_day20/scans/slice_0094_266_266_1.50_1.50.png    3
./train/case101/case101_day20/scans/slice_0095_266_266_1.50_1.50.png    3
...
```

```
./train/case92/case92_day0/scans/slice_0085_266_266_1.50_1.50.png       3
./train/case92/case92_day0/scans/slice_0086_266_266_1.50_1.50.png       3
./train/case92/case92_day0/scans/slice_0087_266_266_1.50_1.50.png       3
./train/case92/case92_day0/scans/slice_0088_266_266_1.50_1.50.png       3
./train/case92/case92_day0/scans/slice_0089_266_266_1.50_1.50.png       3
Name: id, Length: 3201, dtype: int64
```

We conclude there are 3201 images having all 3 segmentation labels.

Lines 94-104

The sklearn.model_selection.GroupKFold is used to create 5 folds. The choice of GroupKFold is done to avoid data leakage. Specifically, each subject has multiple scan sets, these occur on different days. GroupKFold ensures that a subject's data is either in training *or* validation dataset, not both.

The file names are written to "./mmseg_train/splits/fold_N.txt and "./mmseg_train/splits/holdout_N.txt"

Lines 106-118

Here the author is establishing sets of image slices where segmentation is, or is not, present. Sets are established for tails, noanno and faults.

- "end_slice" is the last annotated slice
- "tail_group" is the set of 5 slices immediately following the last annotated slice
- "noanno_group" is the set of slices without annotation
- Set "tails" is a list of image filenames from the "tail_group"
- Set "noannos" is a list of image filenames from the "noanno_group"

An exception is made for "case7_day0" and "case81_day30" (unclear why) - these are added to the "faults" set.

Lines 120-140

The author creates folds for "notail", "noanno", "case". To do this they iterate through the 5 splits. Read the split/fold_N.txt file into a DataFrame. From the first column (case) is created a list, and then the "tails", "noannos" and "faults" are removed from that. Similarly is done the holdout set. The results are written to:

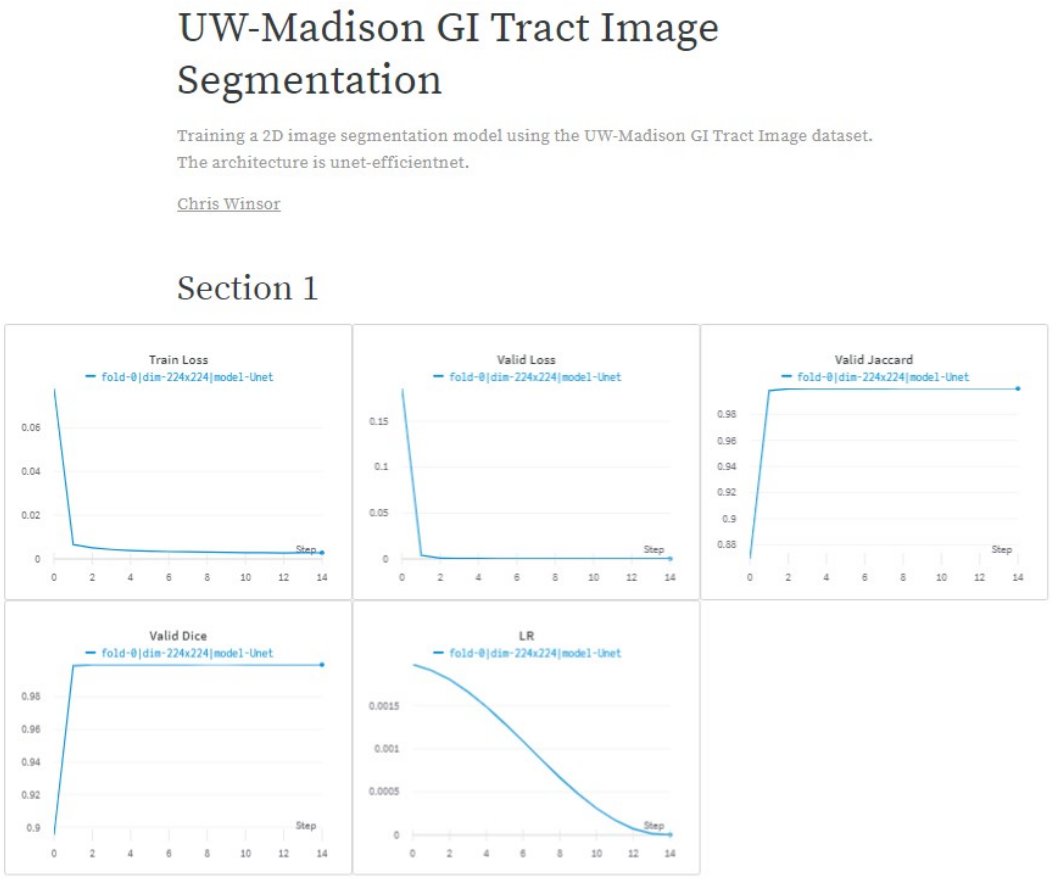
- `mmseg_train/splits_notail/fold_N.txt`
- `mmseg_train/splits_noanno/fold_N.txt`
- `mmseg_train/splits_case/fold_N.txt`
- `mmseg_train/splits_notail/holdout_N.txt`
- `mmseg_train/splits_noanno/holdout_N.txt`
- `mmseg_train/splits_case/holdout_N.txt`

Finally the folds and holdouts are "catted" to

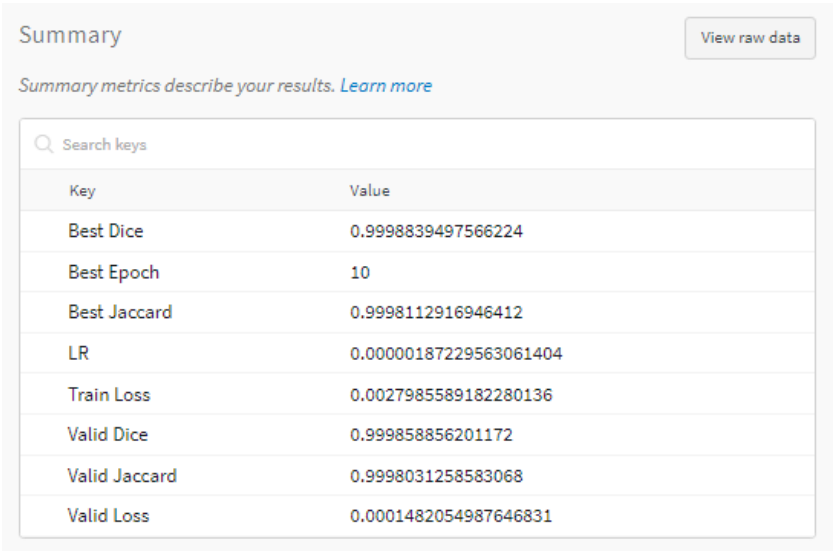
- `mmseg_train/splits["", "_notail", "_noanno", "_case"]/holdout_N.txt`

Results

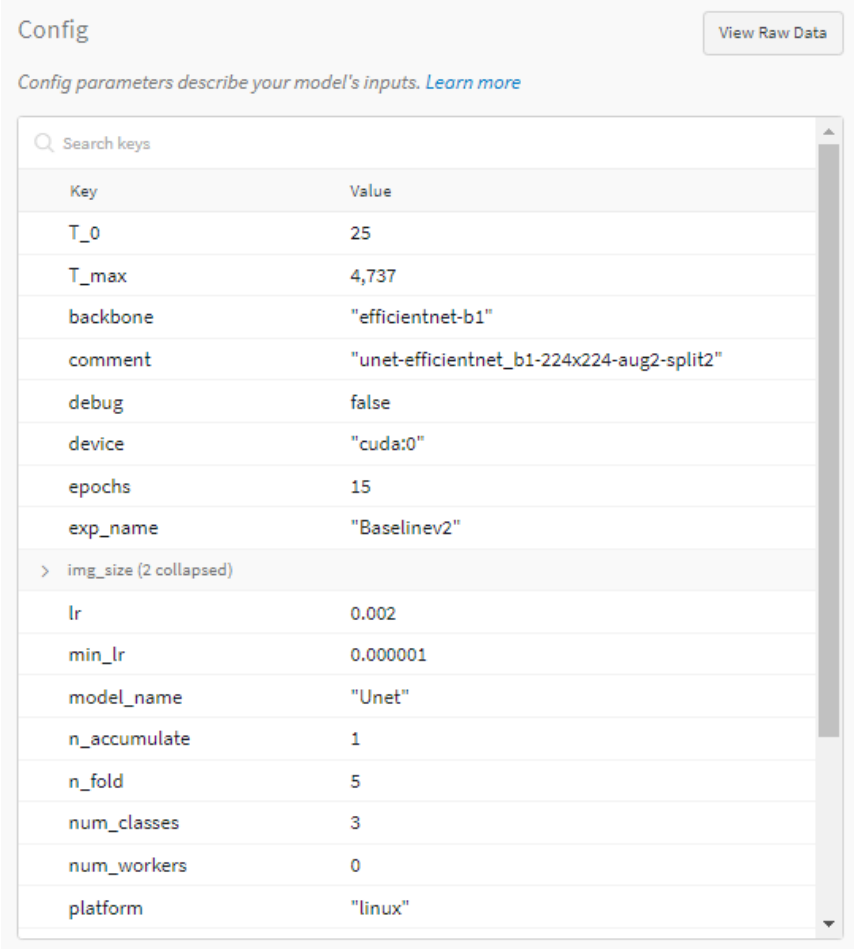
Training was about 6 hours with results captured to WandB. The model quickly stabilized to >98% Jaccard, and finished with .9998%.



.



.



Appendices

Perspective (training a backbone...)

Above project leverages transfer learning (pretrained backbone). Here is some context on what goes into that backbone...

- 27 hours on (8) P100s
- Total GPU memory = (14.5*8)
- ImageNet-5K

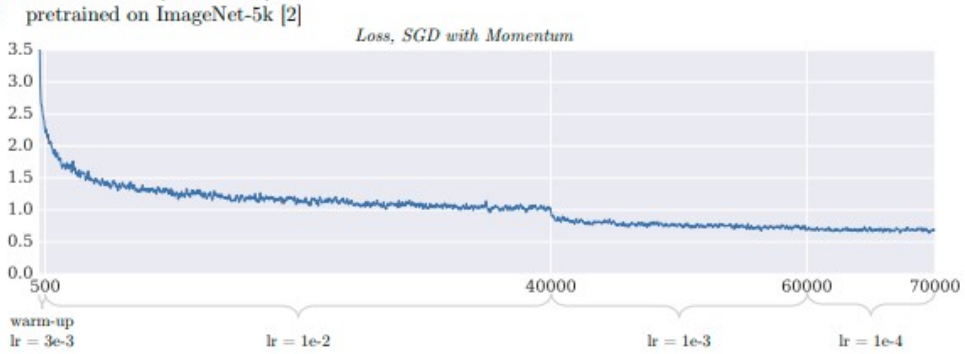
ResNeXt-FPN Training Details

Data augmentation:

Scaling: 0.5x – 2.0x | Crop: 0.9x – 1.0x (max 800x800) | Color augmentation [1]

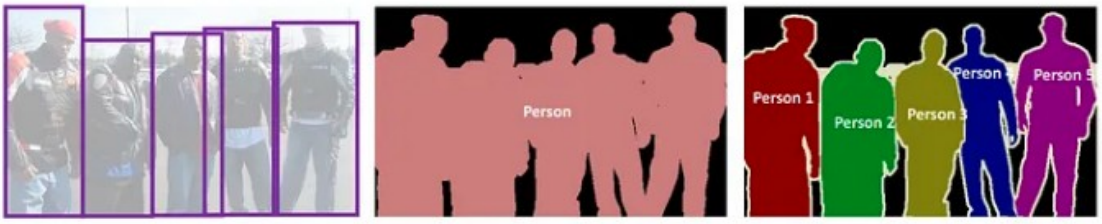
Train stats:

- training time: 27 hours (8 P100 GPUs)
- per GPU memory usage: 14.5GB
- batch size: 16 (2 x 8 GPU)
- pretrained on ImageNet-5k [2]



[1] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. SSD: Single shot multibox detector. ECCV 2016.
[2] Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. Aggregated residual transformations for deep neural networks. CVPR 2017.

Detection, Segmentation



Object Detection Semantic Segmentation **Instance Segmentation**

Liu, Y. (2020, May 18). The Confusing Metrics of AP and mAP for Object Detection. Medium. https://yanfengliux.medium.com/the-confusing-metrics-of-ap-and-map-for-object-detection-3113ba0386ef

Multiple Instances of Class, Adjacent
(U-Net)

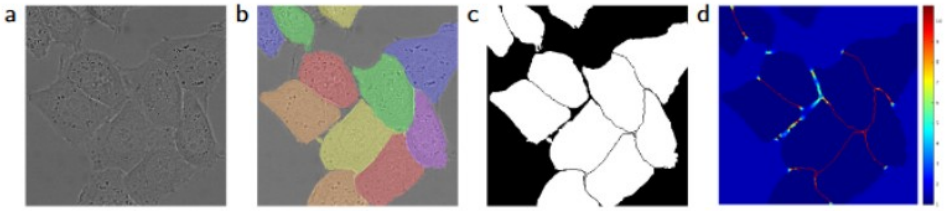


Fig. 3. HeLa cells on glass recorded with DIC (differential interference contrast) microscopy. (**a**) raw image. (**b**) overlay with ground truth segmentation. Different colors indicate different instances of the HeLa cells. (**c**) generated segmentation mask (white: foreground, black: background). (**d**) map with a pixel-wise loss weight to force the network to learn the border pixels.

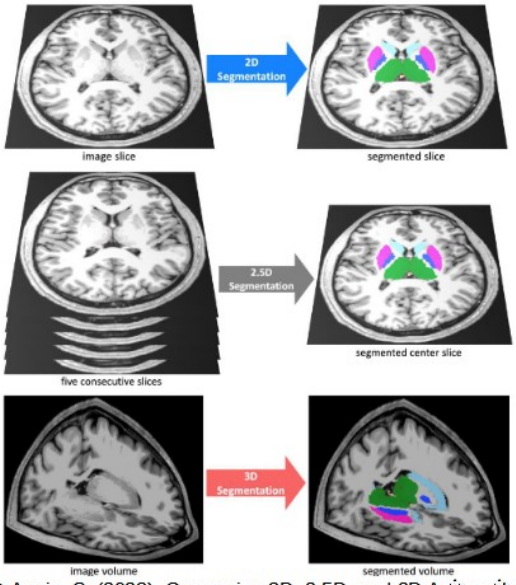
Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation (arXiv:1505.04597). arXiv. https://doi.org/10.48550/arXiv.1505.04597

2D, 2.5D, 3D Models

2D = single MRI slice (64x64 pixels in and out)

2.5D = 64x64x5 (5 slices) predicting segmentation on #3

3D = input and output are 64x64x64 voxels



Avesta, A., Hossain, S., Lin, M., Aboian, M., Krumholz, H. M., & Aneja, S. (2023). Comparing 3D, 2.5D, and 2D Approaches to Brain Image Auto-Segmentation. Bioengineering, 10(2), 181. https://doi.org/10.3390/bioengineering10020181

IoU, Dice

IoU and Dice are common performance metrics for image segmentation. IoU is intersection over union (simply counting number of pixels both in predicted and truth, over the total combined set of pixels in both predicted and truth). Dice is intersection over total number of pixels in the patch.

Precision, Recall

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Liu, Y. (2020, May 18). The Confusing Metrics of AP and mAP for Object Detection. Medium. https://yanfengliux.medium.com/the-confusing-metrics-of-ap-and-map-for-object-detection-3113ba0386ef

P/R Curve

```python
# precision-recall curve and f1
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)
# predict probabilities
lr_probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# predict class values
yhat = model.predict(testX)
lr_precision, lr_recall, _ = precision_recall_curve(testy, lr_probs)
lr_f1, lr_auc = f1_score(testy, yhat), auc(lr_recall, lr_precision)
# summarize scores
print('Logistic: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
# plot the precision-recall curves
no_skill = len(testy[testy==1]) / len(testy)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(lr_recall, lr_precision, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

[2] ✓ 3.6s Python