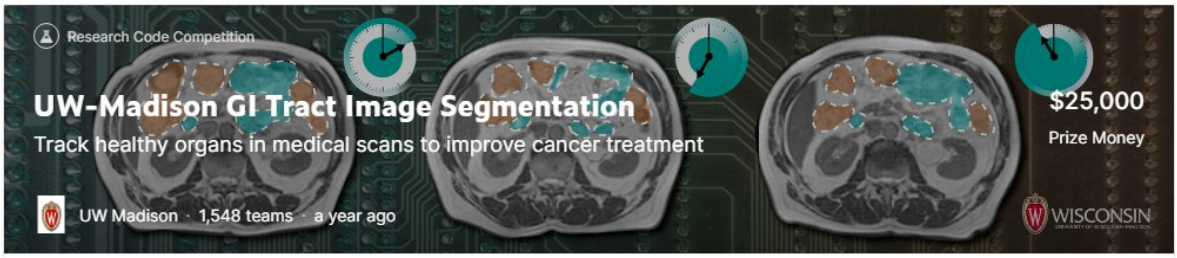
Image Segmentation - Current Techniques for Medical Applications



Chris Winsor
7/24/2023

This is a review of current tools and techniques for image segmentation in medical applications. Our review is driven by the "UW-Madison GI Tract Image Segmentation" competition on Kaggle. We review submissions of top competitors, drill down into details of their code, and stage for benchmark experiments. In the process we uncover powerful and practical techniques, tools, and models.

The UW-Madison 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

Related Studies

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

(Avesta et al., 2023) compares 2D, 2.5D, 3D segmentation in brain imaging.
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

Capsule Network, UNet, nnUNet are evaluated for segmentation of 3 structures in human brain.

Dataset:

- 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)

Training metric is Dice score.

Performance evaluations:

- 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.

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. "

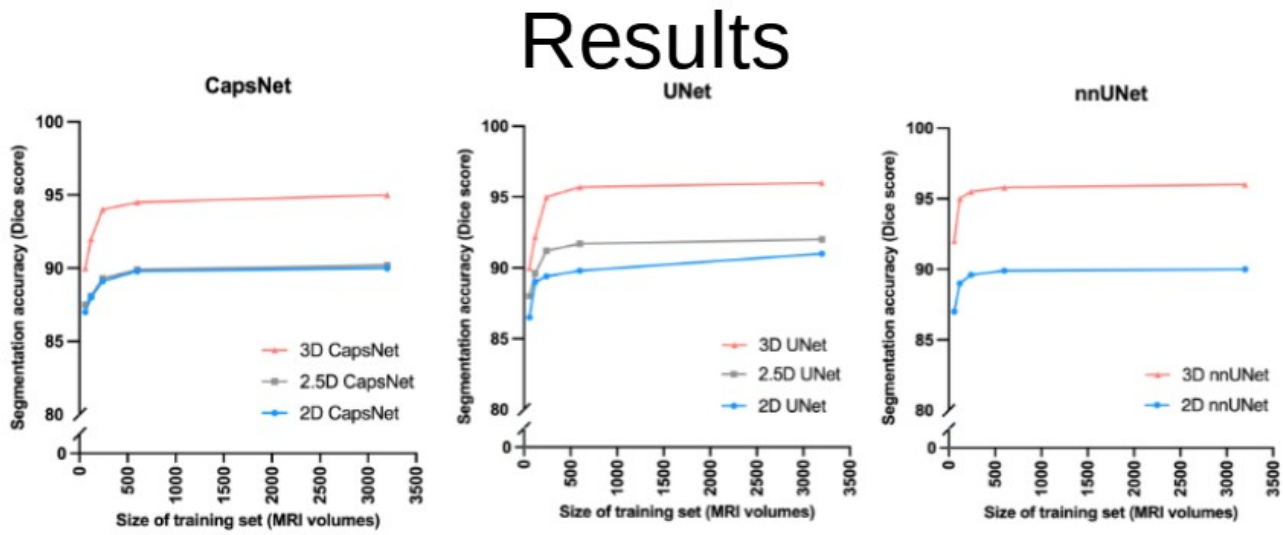


Figure 3. Comparing 3D, 2.5D, and 2D approaches when training data is limited. As we decreased the size of the training set from 3000 MRIs down to 60 MRIs, the CapsNet (**a**), UNet (**b**), and nnUNet (**c**) models maintained higher segmentation accuracy (measured by Dice scores).

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

Commonly Used Tools and Techniques

The following is a short-list of techniques and tool used by top contestants.

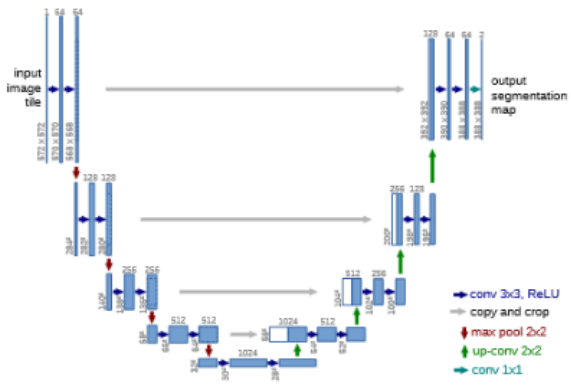
U-Net

U-Net is 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 (segmentation).

Reference: https://en.wikipedia.org/wiki/U-Net (example applications and implementations available)

Segmentation for Biomedical Images
(U-Net)

- Expansive and upsampling are (essentially) symmetric
- Overlapping tile strategy allows for sub-image prediction
- Heavy use of augmentation (elastic deformation) to accommodate small datasets
- Weighted loss focusing on splitting background labels that split for touching objects (multiple instances of sample class)



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

CapsNet

3D Capsule Networks[2] (2022)are an extension to 2D CapsNets[1].

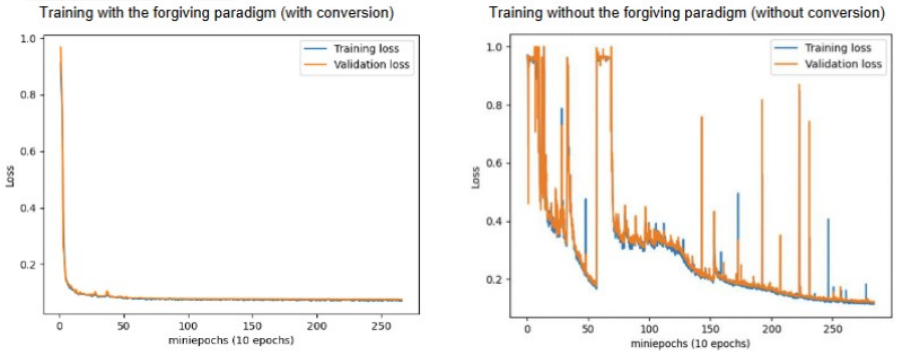
Key novel design elements:

- 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 their github for related work... https://github.com/Aneja-Lab-Yale/Aneja-Lab-Public-3D2D-Segmentation

[1] 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
[2] 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

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) and eliminates unnecessary elements. 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.

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:

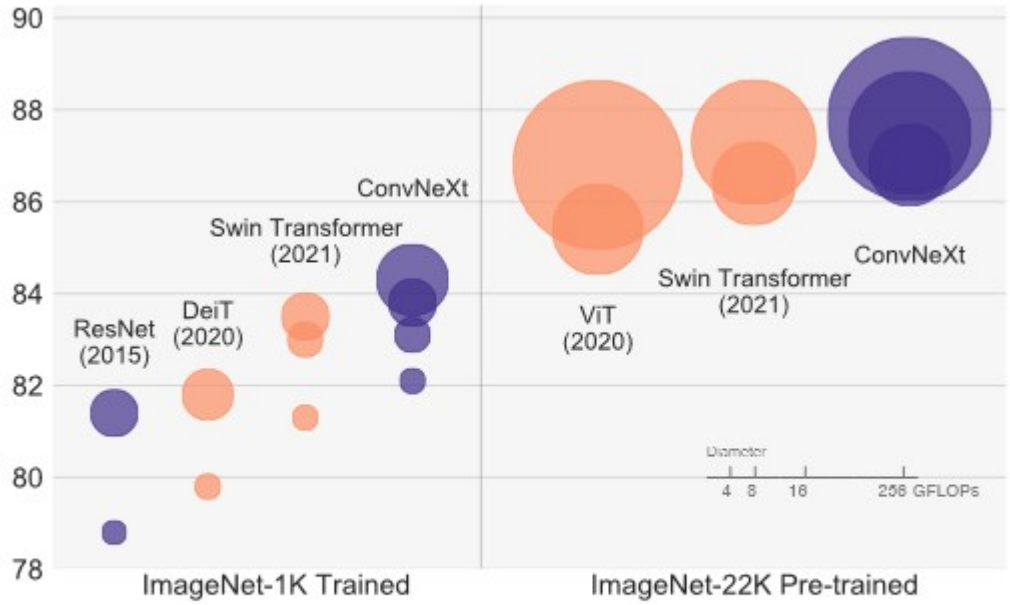


Figure 1. ImageNet-1K classification results for ● ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family.

Segmentation

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**

EfficientNet

EfficientNet is introduced by Tan et al. (2020). It 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.
- EfficientNet is also 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."

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

Swin Transformer (Shifted Windows Transformer)

Liu et al (Microsoft Research Asia) 2021

Swin 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

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

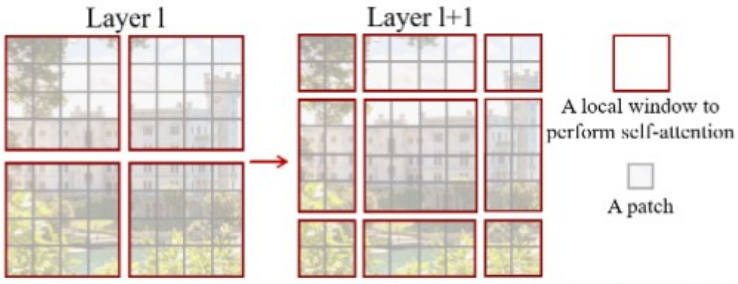


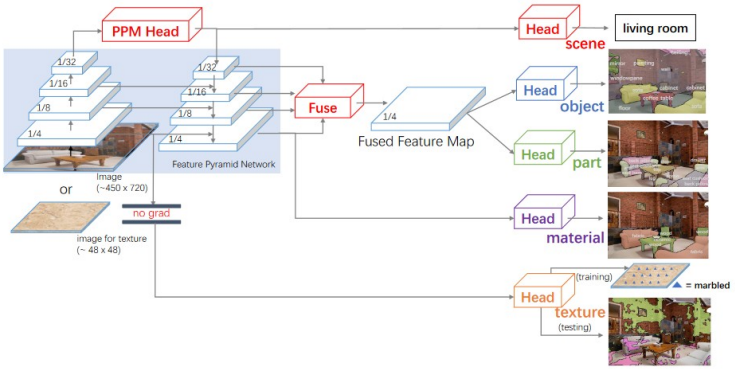
Figure 2. An illustration of the *shifted window* approach for computing self-attention in the proposed Swin Transformer architecture. In layer l (left), a regular window partitioning scheme is adopted, and self-attention is computed within each window. In the next layer $l + 1$ (right), the window partitioning 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.

UPerNet (Unified Perceptual Parsing) (Xiao et al. 2018)

References:
https://arxiv.org/abs/1807.10221
https://huggingface.co/docs/transformers/main/model_doc/upernet

Contributions:
Unified Perceptual Parsing (UPP) "task" - parsing multiple visual concepts at once
UPerNet network architecture

UPerNet predicts (models) features various complexity and scope - texture, material, object, part, scene
Is paired with a FPN (Feature Pyramid Network) or similar
UPerNet takes input from the FPN's intermediate layers.



MONAI

https://github.com/Project-MONAI/MONAI
The MONAI Model Zoo (https://github.com/Project-MONAI/model-zoo) is a set of medical imaging models.

MMsegmentation end-to-end notebook, Segmentation Models (PyTorch)

The discussion provides a baseline notebook for UW-Madison based on MMsegmentation (mmseg) framework. The baseline uses UNet, EfficientNet models and approach.
References:
https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/discussion/323921
https://segmentation-modelspytorch.readthedocs.io/en/latest/
https://mmsegmentation.readthedocs.io/en/latest/api.html

The discussion references:
"segmentation_models_pytorch" (Torch - reference below)
"mmseg" (API for segmentation / hyperparameter tuning)

CODE:
[Notebook] https://www.kaggle.com/code/carnozhao/uwmgit-mmsegmentation-end-to-end-submission
[Github repo] https://gitthub.com/CarnoZhao/Kaggle-UWMGIT

Submission Detailed Reviews

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

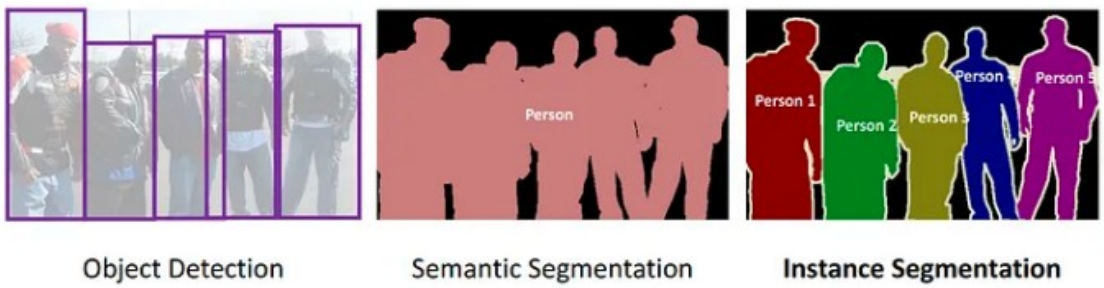
Code available: No

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

Appendices

Detection, 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)

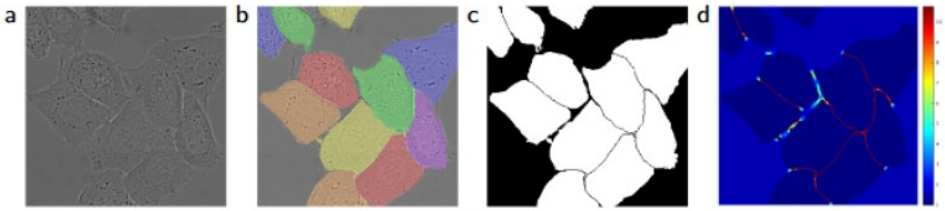


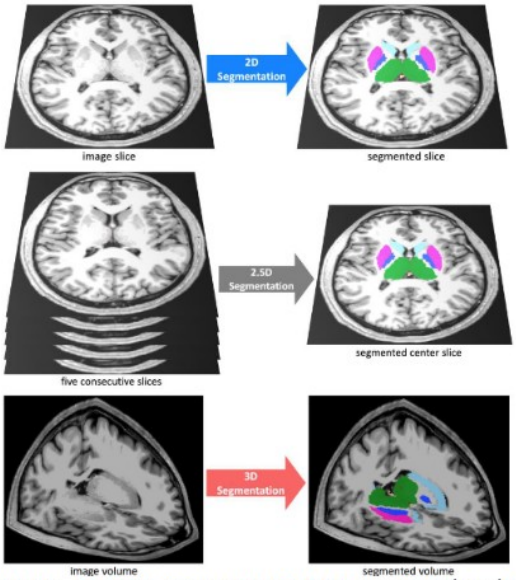
Fig. 3. HeLa cells on glass recorded with DIC (differential interference contrast) microscopy. (**a**) raw image. (**b**) overlay with ground truth segmentation. Different colors indicate different instances of the HeLa cells. (**c**) generated segmentation mask (white: foreground, black: background). (**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 = 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

2D, 2.5D, 3D Models

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

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

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

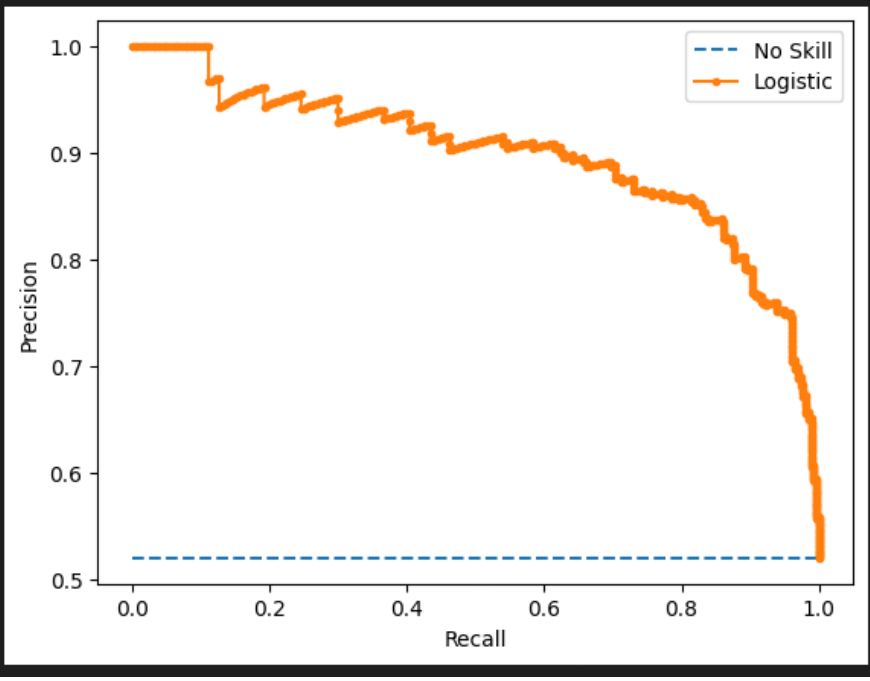
		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

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



Experiments

01_cars

Following "cars segmentation" example in Segmentation Models PyTorch

https://github.com/qubvel/segmentation_models.pytorch/blob/master/examples/cars%20segmentation%20(camvid).ipynb

Dataset (CamVid) General Info

- http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/
- The dataset is 10 minutes of 30hz video with semantic labels at 1hz and in part 15hz.
- per-pixel segmentation of 700 images (curated)
- high resolution color video
- 3D pose for each frame (using calibrated camera intrinsics)
- Labeling software

Dataset References

- (1) Segmentation and Recognition Using Structure from Motion Point Clouds, ECCV 2008 (pdf) Brostow, Shotton, Fauqueur, Cipolla (bibtex)
- (2) Semantic Object Classes in Video: A High-Definition Ground Truth Database (pdf) Pattern Recognition Letters (to appear) Brostow, Fauqueur, Cipolla (bibtex)

Benchmark Parameters

Dataset As Used:

- Image size = 320x480
- Image type: RGB .PNG
- 32 classes
- 701 images total. (Train, Val, Test) = (367, 101, 233)

Model/Architecture:

- FPN (feature pyramid network)
- Encoder = se_resnext50_32x4d (25M parameters)
- Encoder weights (pretraining) = "imagenet"

Training:

- GPU: 11 seconds/epoch (used 4GB mem)

- CPU: 110 seconds/epoch
- After 2 epochs:
  - dice loss = 0.19
  - iou score = 0.68

Visualization after 2 epochs training (~2minutes)

- Segmentation is blob-like but clearly working.

Runtime:

On test set (233 images, with augmentation)

- GPU: 11.6 seconds (21.2 img/sec = 0.05 sec/img) - GPU used 4GB, also this used 40% CPU
- CPU: 40 seconds (5.8 img/sec = 0.17 sec/img) - this used 100% CPU

02_uw_madison

Benchmark Parameters

Dataset:

- Image: 310x360, 16 bit grayscale.
- Annotation: RLE-encoded mask
- Each case may show up on several days (preprocessing implication -> avoid leakage)
  - train: 38,496 images, each with 3 segmentation classes (stomach, small_bowel, large_bowl)
- 4.7GB (total)

To Give Perspective (training a backbone...)

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