# Best Research Practices

Dong Yang
NVIDIA

# Medical Image Analysis

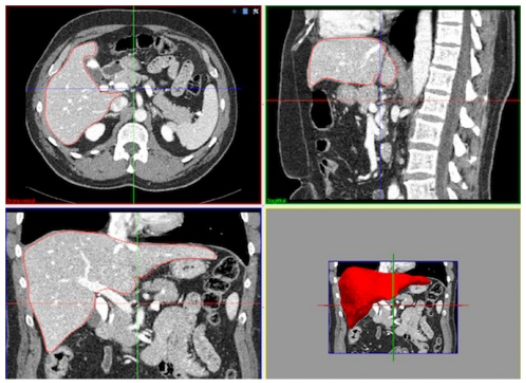
## Motivations

- *What?*
  - *To gain high-level understanding from medical images*
- *Why?*
  - *Disease diagnosis, treatment planning and surgical guidance*
  - *Applications*
    - *Precise radiotherapy (calculate dose distribution, delineate tumours and normal organs)*
    - *"COVID-19" screening, etc.*

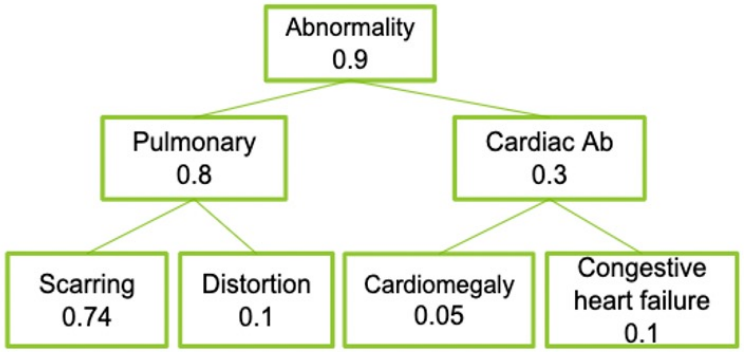
# Medical Image Analysis

## Applications

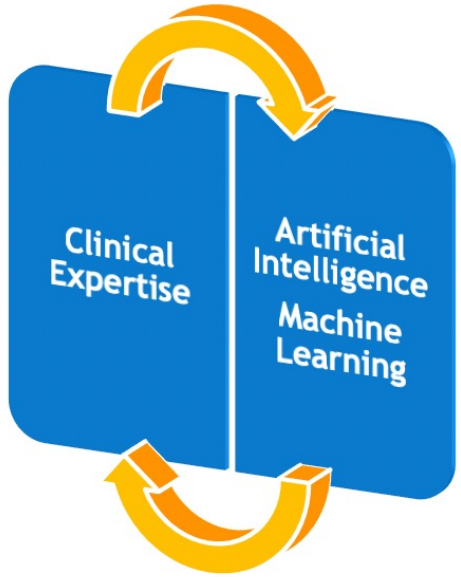
CT
MRI
Ultrasound
X-Ray
Microscopy
Pathology
PET
OCT
EHR

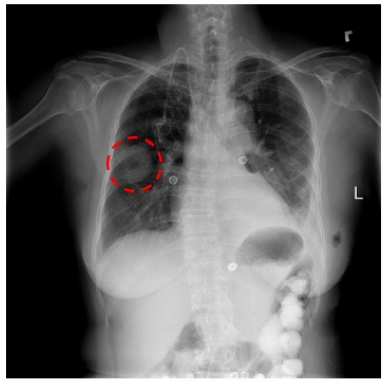


**Segmentation**
Anatomy Understanding

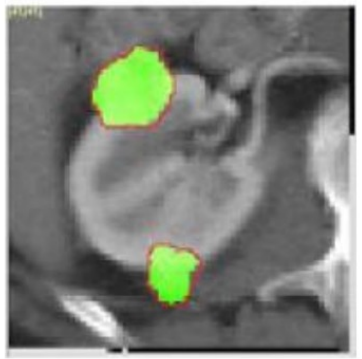



**Classification**
Abnormality Detection

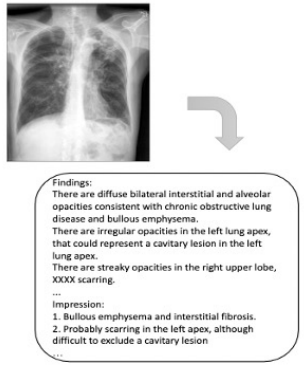


**Detection**
Cancer Staging



**Survival Model Prediction**
Longitudinal Monitoring



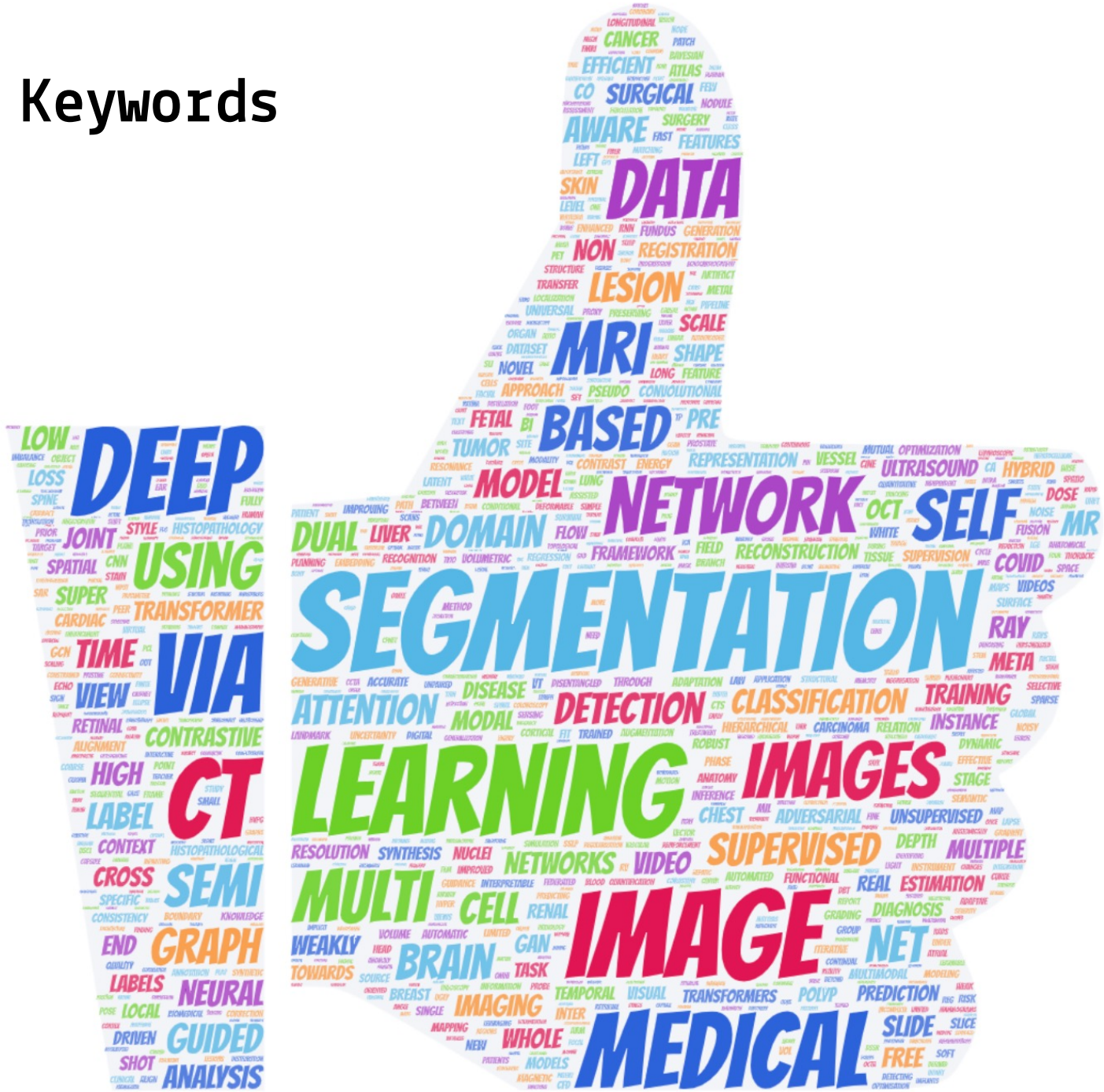

**Natural Language Processing**
Medical Report

# MICCAI 2021 Keywords

*Segmentation*

*Deep*

*Learning*

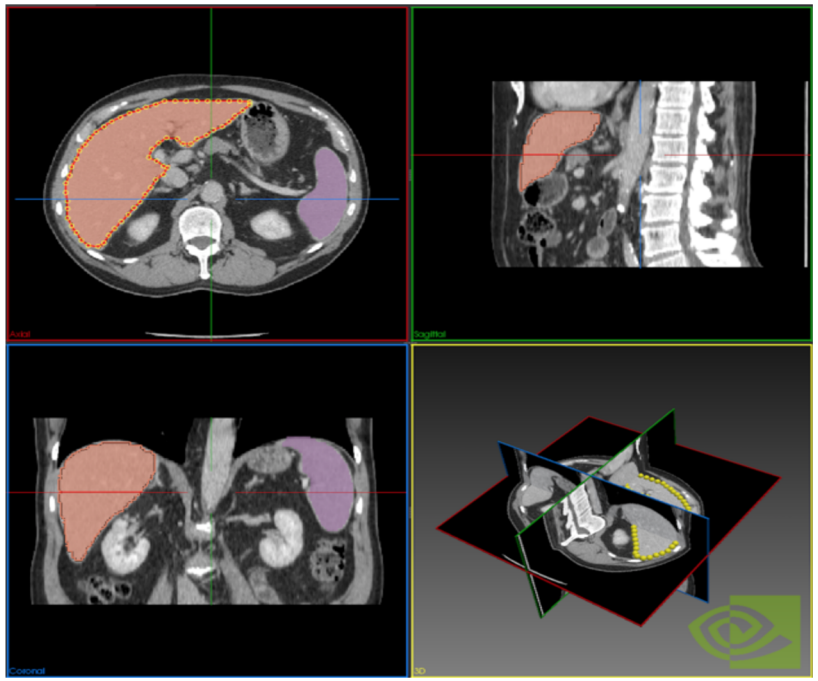


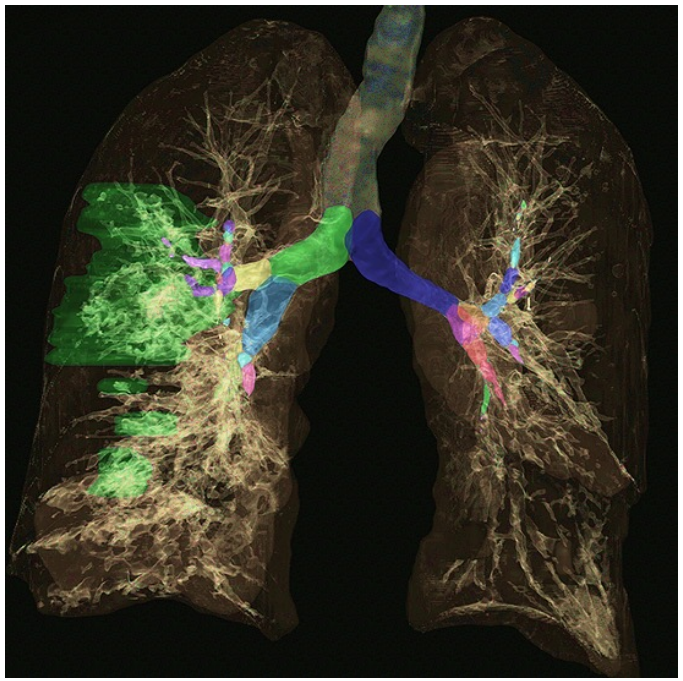

https://wordart.com/

# Case Study

## 3D Medical Image Segmentation

*Taking 3D medical images (e.g., CT, MRI) as input to extract 3D structure of organs or tumours*





https://devblogs.nvidia.com/annotation-transfer-learning-clara-train/
https://www.nvidia.com/en-us/clara/

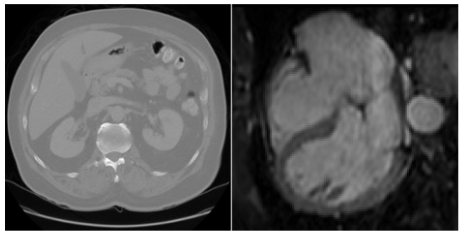
# Applications and Algorithms

## Model Training

*What is the optimal deep learning solution in medical imaging?*

**Data**

*Image*
CT/MRI/Histopathology



*Label*
Classes/Segmentation

**Data Processing**

*Data Pre-Processing*
- Intensity clipping
- Normalization
- Re-sampling

*Data Augmentation*
- Random cropping
- Random flipping
- Random rotation

**Neural Network**

*Classification*
- EfficientNet
- ResNet

*Segmentation*
- AH-Net
- ResUNet
- U-Net
- V-Net

**Hyper-Parameters**

*Optimization*
- Iteration number
- Mini-batch size
- Patch size
- Learning rate
- Learning rate policy
- Sliding-window size
- Weight decay

# Large-Scale Medical Image Segmentation

## Challenges

- *Diversity: Various clinical applications*
  - *Different anatomical objects in different image modalities*
- *Hard to collect large-scale pixel-level annotation*
  - *e.g., takes hours or days to accomplish multi-organ annotation in whole-body CT*
- *Large GPU computation resource request*
  - *High dimensional medical images*

MONAI

# Solutions? Yes

# Case Study

## 3D Spleen CT Segmentation

*41 abdominal CT volumes from public domain (Medical Segmentation Decathlon) with spleen region annotation*

- *32 for training*
- *9 for validation*

*“State-of-the-art” performance around Dice’s score 0.95*

*Publicly available implementation takes several hours to train segmentation models and reach “state-of-the-art” performance*

# Case Study

## 3D Spleen CT Segmentation

*41 abdominal CT volumes from public domain (Medical Segmentation Decathlon) with spleen region annotation*

- *32 for training*
- *9 for validation*

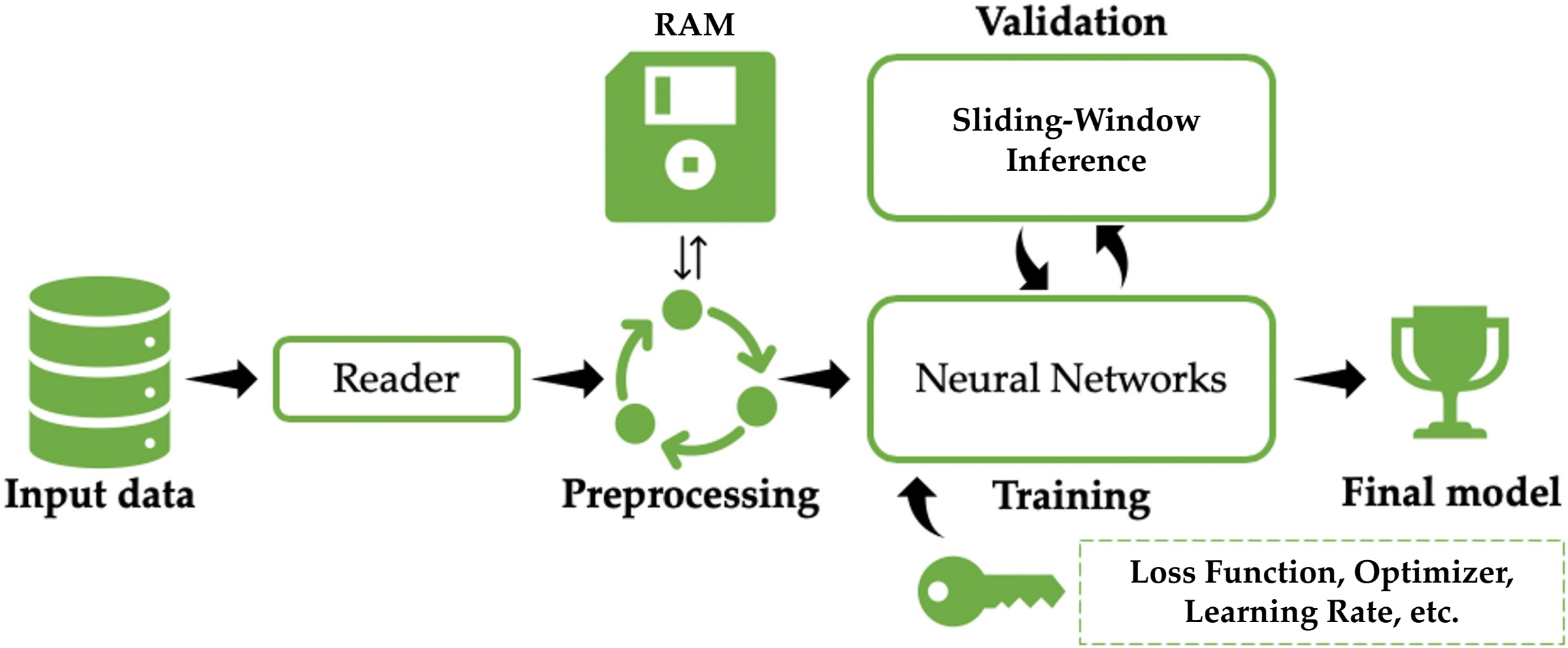
*"State-of-the-art" performance around Dice's score 0.95*

*Publicly available implementation takes several hours to train segmentation models and reach "state-of-the-art" performance*

*But now, we can achieve it within **one minute** using U-Net on one single GPU*

# Case Study

## 3D Spleen CT Segmentation



# 3D Spleen CT Segmentation

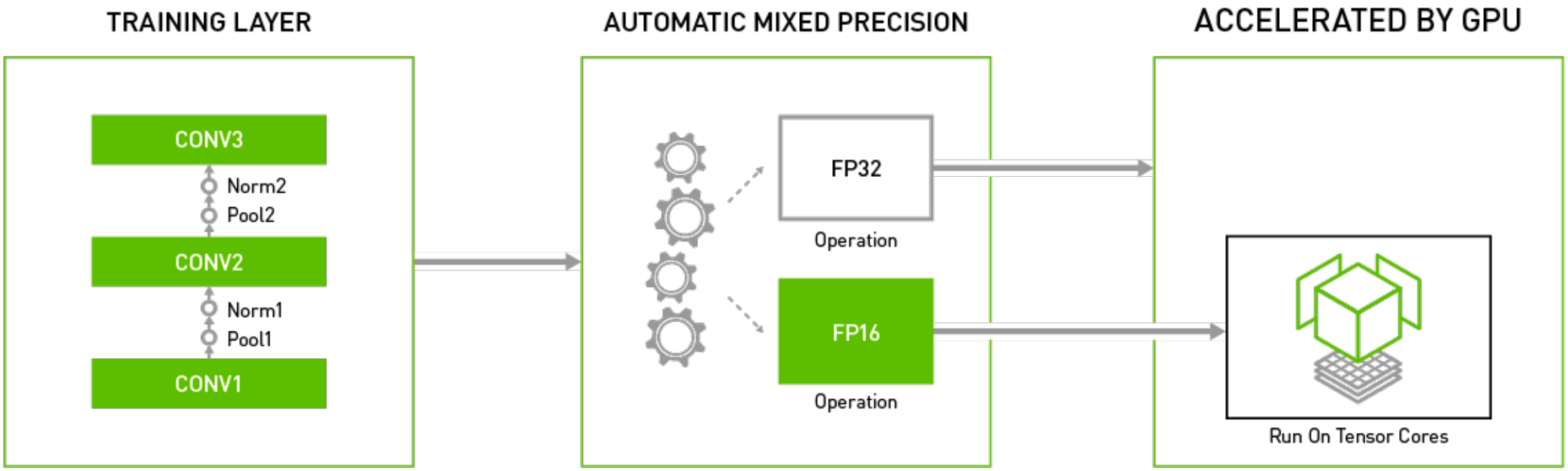
## Pre-Processing and Augmentation

- *“Spacingd”*
  - *Resampling | Physical meanings*
- *“Orientationd”*
  - *Aligning imaging origins*
- *“ScaleIntensityRanged”*
  - *Intensity window | Hounsfield units (HU)*
- *“RandCropbyPosNegLabeld”*
  - *Handling various image size*
  - *Balancing training samples*

```
train_transforms = Compose(
    [
        LoadImaged(keys=["image", "label"]),
        EnsureChannelFirstd(keys=["image", "label"]),
        Spacingd(keys=["image", "label"], pixdim=(
            1.5, 1.5, 2.0), mode=("bilinear", "nearest")),
        Orientationd(keys=["image", "label"], axcodes="RAS"),
        ScaleIntensityRanged(
            keys=["image"], a_min=-57, a_max=164,
            b_min=0.0, b_max=1.0, clip=True,
        ),
        CropForegroundd(keys=["image", "label"], source_key="image"),
        RandCropByPosNegLabeld(
            keys=["image", "label"],
            label_key="label",
            spatial_size=(96, 96, 96),
            pos=1,
            neg=1,
            num_samples=4,
            image_key="image",
            image_threshold=0,
        ),
        # user can also add other random transforms
        # RandAffined(
        #     keys=['image', 'label'],
        #     mode=('bilinear', 'nearest'),
        #     prob=1.0, spatial_size=(96, 96, 96),
        #     rotate_range=(0, 0, np.pi/15),
        #     scale_range=(0.1, 0.1, 0.1)),
        EnsureTyped(keys=["image", "label"]),
    ]
)
```

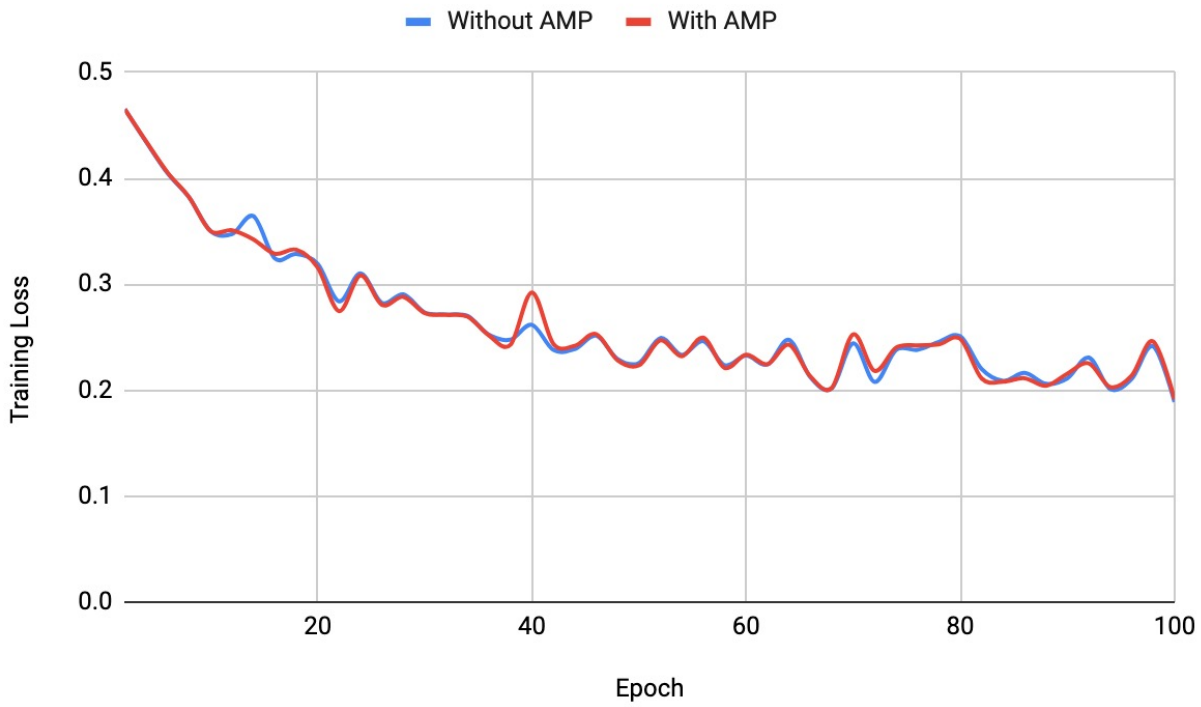
# 3D Spleen CT Segmentation

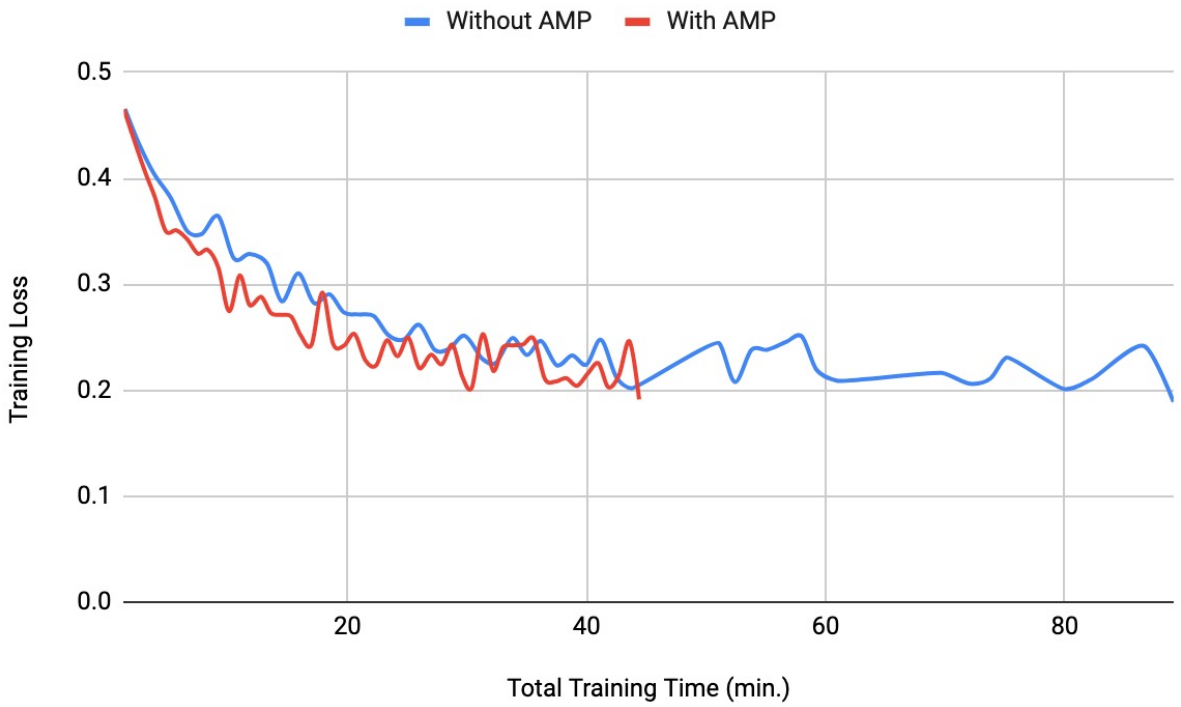
## Automated Mixed Precision (AMP)



# 3D Spleen CT Segmentation

## Automated Mixed Precision (AMP)



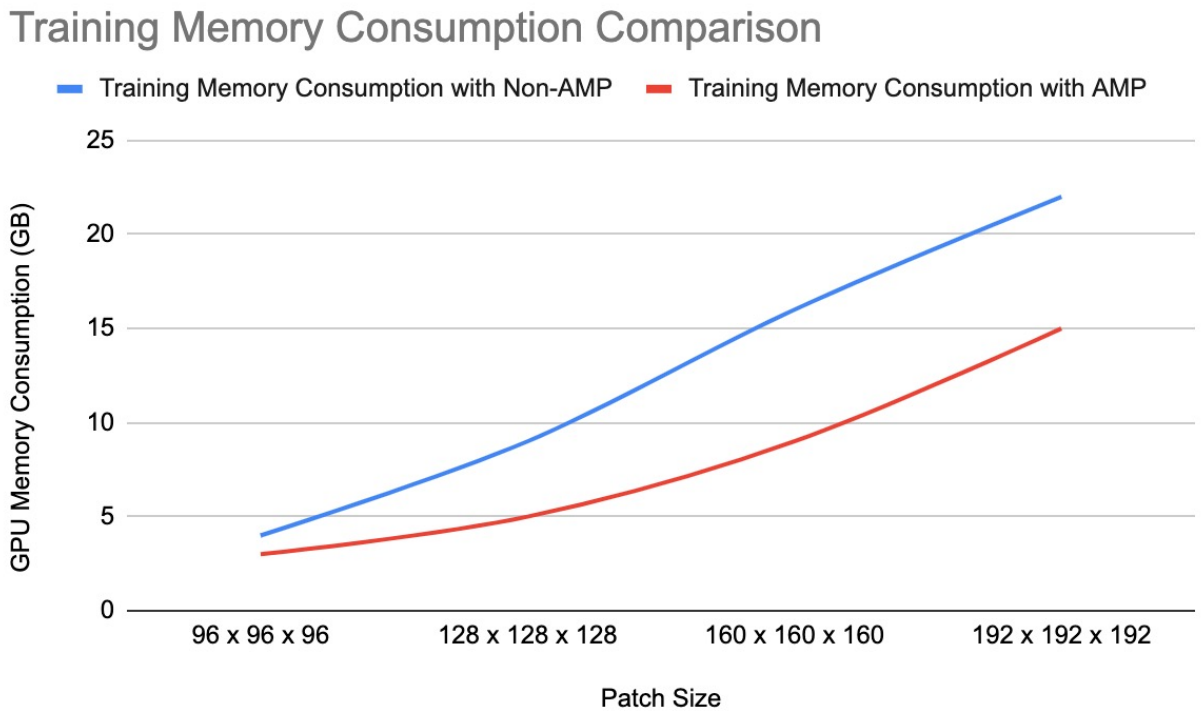
# 3D Spleen CT Segmentation

## Automated Mixed Precision (AMP)

| Method | Total | Forward | Backward | Loss | Optimizer |
|---|---|---|---|---|---|
| Training Patch Size 128 x 128 x 128 \| **Batch Size 2** | | | | | |
| Non-AMP | 0.362 s | 0.081 s | 0.252 s | 0.001 s | 0.028 s |
| AMP | 0.181 s | 0.062 s | 0.091 s | 0.001 s | 0.027 s |
| Speed-up | x 2.0 | x 1.3 | x 2.8 | x 1.0 | x 1.0 |
| Training Patch Size 128 x 128 x 128 \| **Batch Size 8** | | | | | |
| Non-AMP | 1.381 s | 0.340 s | 1.009 s | 0.002 s | 0.030 s |
| AMP | 0.515 s | 0.150 s | 0.337 s | 0.002 s | 0.026 s |
| Speed-up | x 2.7 | x 2.3 | x 3.0 | x 1.0 | x 1.2 |

# 3D Spleen CT Segmentation

## Automated Mixed Precision (AMP)



# 3D Spleen CT Segmentation

## Novograd Optimizer



### Training Deep Networks with Stochastic Gradient Normalized by Layerwise Adaptive Second Moments

**Boris Ginsburg** [1] **Patrice Castonguay** [1] **Oleksii Hrinchuk** [1] **Oleksii Kuchaiev** [1] **Ryan Leary** [1] **Vitaly Lavrukhin** [1] **Jason Li** [1] **Huyen Nguyen** [1] **Yang Zhang** [1] **Jonathan M. Cohen** [1]

**Abstract**

We propose NovoGrad, an adaptive stochastic gradient descent method with layer-wise gradient normalization and decoupled weight decay. In our experiments on neural networks for image classification, speech recognition, machine translation, and language modeling, it performs on par or better than well-tuned SGD with momentum, Adam, and AdamW. Additionally, NovoGrad (1) is robust to the choice of learning rate and weight initialization, (2) works well in a large batch setting, and (3) has half the memory footprint of Adam.

### 1. Introduction

The most popular algorithms for training Neural Networks (NNs) are Stochastic Gradient Descent (SGD) with momentum (Polyak, 1964; Sutskever et al., 2013) and Adam (Kingma & Ba, 2015). SGD with momentum is the preferred algorithm for computer vision, while Adam is more commonly used for natural language processing (NLP) and speech problems. Compared to SGD, Adam is perceived as safer and more robust to weight initialization and learning rate. However, Adam has certain drawbacks. First, as noted in the original paper (Kingma & Ba, 2015), the second moment can vanish or explode, especially during the initial phase of training. Also, Adam often leads to solutions that generalize worse than SGD (Wilson et al., 2017), e.g.

We started with Adam, and then (1) replaced the element-wise second moment with the layer-wise moment, (2) computed the first moment using gradients normalized by layer-wise second moment, (3) decoupled weight decay (WD) from normalized gradients (similar to AdamW).

The resulting algorithm, *NovoGrad*, combines SGD's and Adam's strengths. We applied NovoGrad to a variety of large scale problems — image classification, neural machine translation, language modeling, and speech recognition — and found that in all cases, it performs as well or better than Adam/AdamW and SGD with momentum.

### 2. Related Work

NovoGrad belongs to the family of **Stochastic Normalized Gradient Descent (SNGD)** optimizers (Nesterov, 1984; Hazan et al., 2015). SNGD uses only the direction of the stochastic gradient $\boldsymbol{g}_t$ to update the weights $\boldsymbol{w}_t$:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \lambda_t \cdot \frac{\boldsymbol{g}_t}{||\boldsymbol{g}_t||}$$

The step size does not depend on the magnitude of that gradient. Hazan et al. (2015) proved that the direction of the gradient is sufficient for convergence. Ignoring the gradient magnitude makes SNGD robust to vanishing and exploding gradients, but SNGD is still sensitive to "noisy" gradients, especially during an initial training phase.

One can improve SNGD stability by gradient averaging. Adagrad (Duchi et al., 2011), RmsProp (Tieleman & Hin-



Table 2: ImageNet, large batch training comparison: ResNet-50v2, top-1 accuracy(%).

| Optimizer | Reference | Bag of Tricks | Epochs | B=8K | B=32K |
|---|---|---|---|---|---|
| SGD | (Goyal et al., 2017) | warmup | 90 | 76.26 | 72.45 |
| SGD | (You et al., 2018) | warmup, LARS | 90 | 75.30 | 75.40 |
| SGD | (Codreanu et al., 2017) | warmup, multi-step WD | 100 | 76.60 | 75.31 |
| NovoGrad | | — | 90 | 76.64 | 75.78 |
| | | warmup | 90 | — | 75.99 |

ing cosine LR decay. To emulate a large batch, we used a mini-batch of 128 per GPU and accumulated gradients from several mini-batches before each weight update.

To establish the baseline for NovoGrad training with batch 32K we first used the method similar to proposed in Goyal et al. (2017): scaling the learning rate linearly with the batch size and using LR warmup. This method gives top-1=75.09% and top-5=92.27%. We found that we get better results when we increase both the learning rate $\lambda$ and the weight decay $d$ to improve the regularization (see Table 3).

Table 3: ImageNet, large batch training with NovoGrad: ResNet-50v2, 90 epochs. top-1/top-5 accuracy (%).

| Batch | LR | WD | Top-1/Top-5,% |
|---|---|---|---|
| 1K | 0.070 | 0.002 | 76.86/93.31 |
| 8K | 0.016 | 0.006 | 76.64/93.14 |
| 32K | 0.026 | 0.010 | 75.78/92.54 |

For comparison, we took three methods, which (1) use fixed batch size during training and (2) do not modify the original model. All three methods employ SGD with momentum. The first method (Goyal et al. (2017)) scales LR linearly with batch size and uses the LR warmup to stabilize the initial training phase. The second method (You et al. (2018)) combines warmup with Layer-wise Adaptive Rate Scaling (LARS) (You et al., 2017). The last method (Codreanu et al. (2017)) uses warmup and dynamic weight decay (WD). NovoGrad outperformed other methods without any addi-

Error Rates (WER) comparing to SGD, especially for the long runs. The model and training parameters are described in Li et al. (2019).

Table 4: Speech Recognition: Jasper-10x5 trained on LibriSpeech for 400 epochs, greedy WER(%).

| Optimizer | Dev | | Test | |
|---|---|---|---|---|
| | clean | other | clean | other |
| Adam | 5.06 | 15.76 | 5.27 | 15.94 |
| SGD | 4.08 | 13.23 | 4.22 | 13.15 |
| NovoGrad | 3.71 | 11.75 | 3.71 | 11.85 |

#### 4.4. Large Batch Training for Speech Recognition

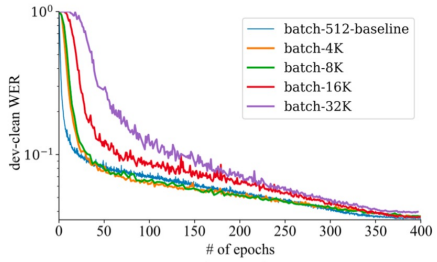



Figure 1: Speech Recognition, large batch training, Jasper-

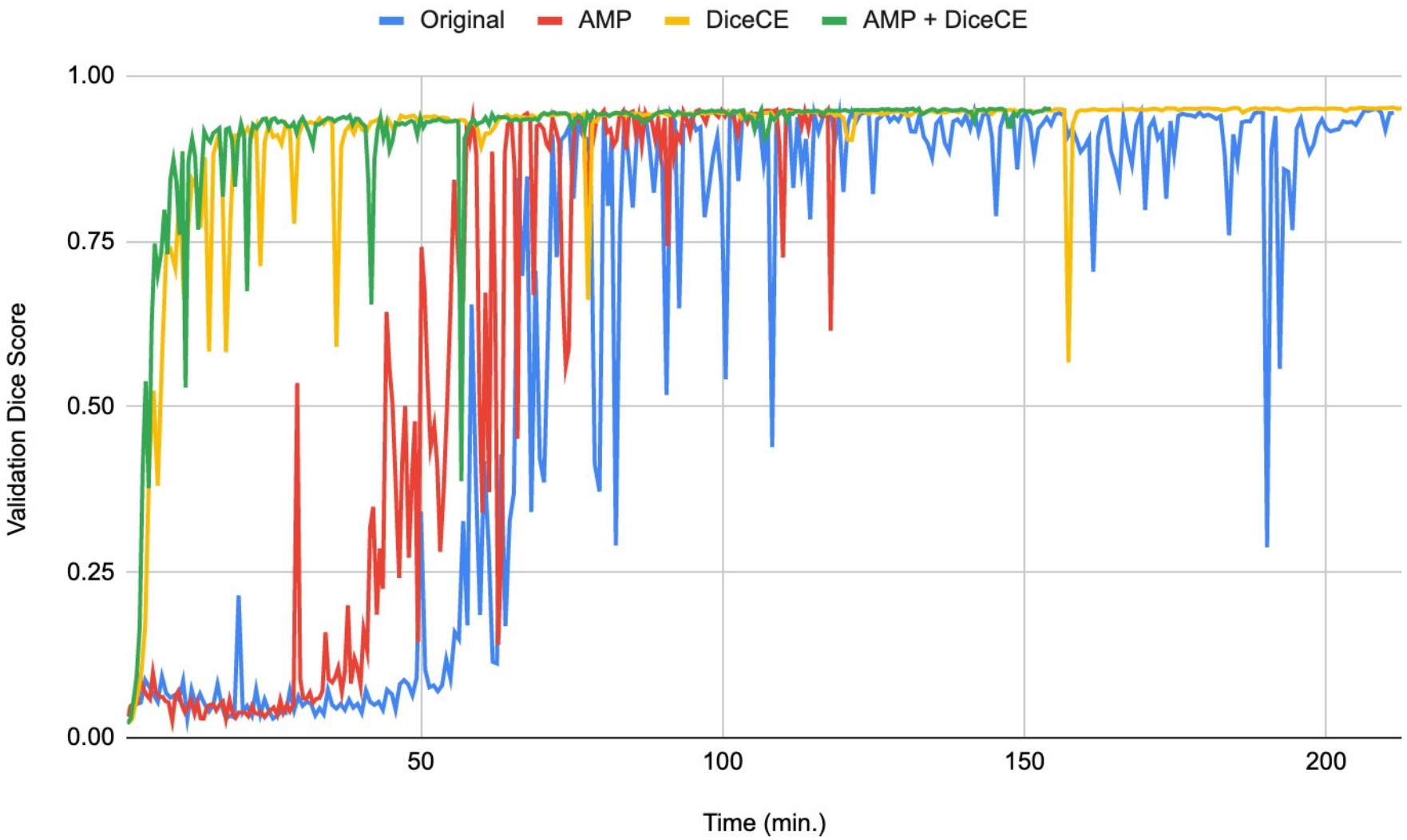
# 3D Spleen CT Segmentation

## Loss function

- *Dice loss = 1 - Dice Score = 1 -* $\frac{2 * |X \cap Y|}{|X| + |Y|}$
- *DiceLoss -> DiceCELoss (dice loss and cross-entropy loss)*
- *“One-line change”:*
- *Before*
  - `loss_function = DiceLoss(to_onehot_y=True, softmax=True)`
- *After*
  - `loss_function = DiceCELoss(to_onehot_y=True, softmax=True, squared_pred=True, batch=True)`

# 3D Spleen CT Segmentation

## Loss function



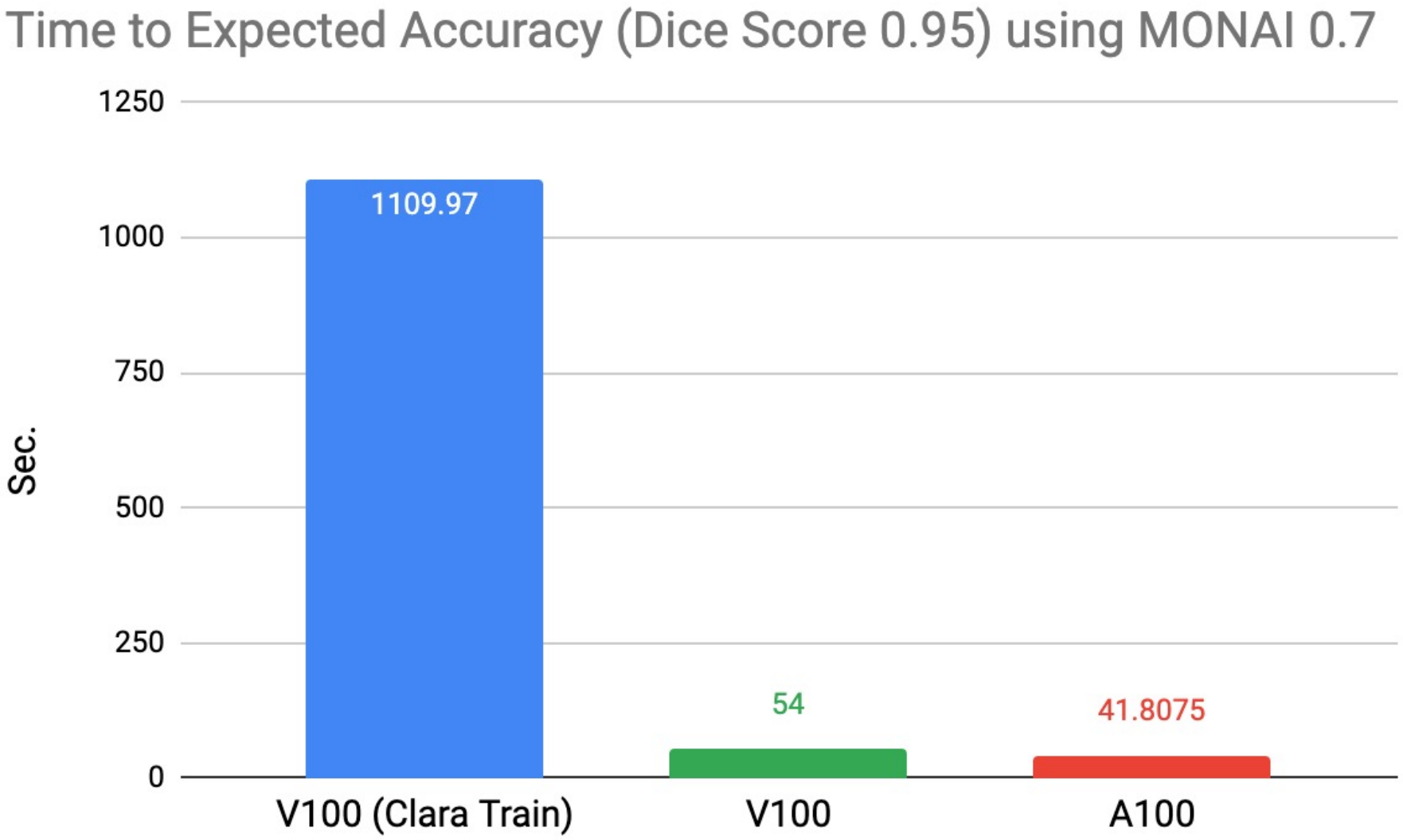
# 3D Spleen CT Segmentation

## Implementation Improvement

- *(fully) GPU based*
  - *Data caching*
  - *Data pre-processing*
  - *Data augmentation*
  - *Sliding-window inference*
  - *etc.*
- *No CPU operations involved*

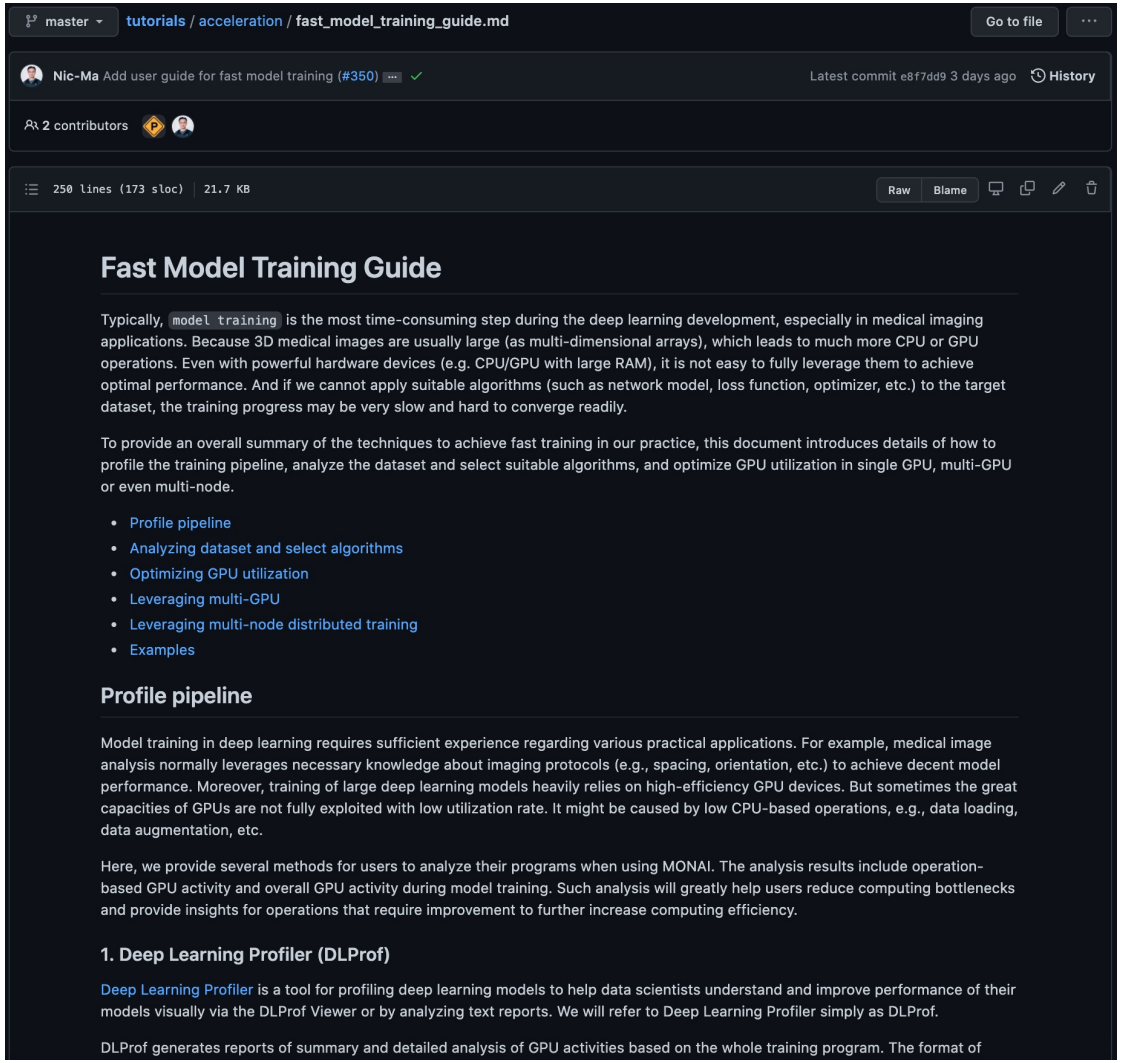
# 3D Spleen CT Segmentation

## Results



# 3D Spleen CT Segmentation

## Tutorial



master tutorials / acceleration / fast_model_training_guide.md Go to file ...

Nic-Ma Add user guide for fast model training (#350) ... ✓ Latest commit e8f7dd9 3 days ago History

2 contributors

250 lines (173 sloc) | 21.7 KB Raw Blame

### Fast Model Training Guide

Typically, `model training` is the most time-consuming step during the deep learning development, especially in medical imaging applications. Because 3D medical images are usually large (as multi-dimensional arrays), which leads to much more CPU or GPU operations. Even with powerful hardware devices (e.g. CPU/GPU with large RAM), it is not easy to fully leverage them to achieve optimal performance. And if we cannot apply suitable algorithms (such as network model, loss function, optimizer, etc.) to the target dataset, the training progress may be very slow and hard to converge readily.

To provide an overall summary of the techniques to achieve fast training in our practice, this document introduces details of how to profile the training pipeline, analyze the dataset and select suitable algorithms, and optimize GPU utilization in single GPU, multi-GPU or even multi-node.

- Profile pipeline
- Analyzing dataset and select algorithms
- Optimizing GPU utilization
- Leveraging multi-GPU
- Leveraging multi-node distributed training
- Examples

### Profile pipeline

Model training in deep learning requires sufficient experience regarding various practical applications. For example, medical image analysis normally leverages necessary knowledge about imaging protocols (e.g., spacing, orientation, etc.) to achieve decent model performance. Moreover, training of large deep learning models heavily relies on high-efficiency GPU devices. But sometimes the great capacities of GPUs are not fully exploited with low utilization rate. It might be caused by low CPU-based operations, e.g., data loading, data augmentation, etc.

Here, we provide several methods for users to analyze their programs when using MONAI. The analysis results include operation-based GPU activity and overall GPU activity during model training. Such analysis will greatly help users reduce computing bottlenecks and provide insights for operations that require improvement to further increase computing efficiency.

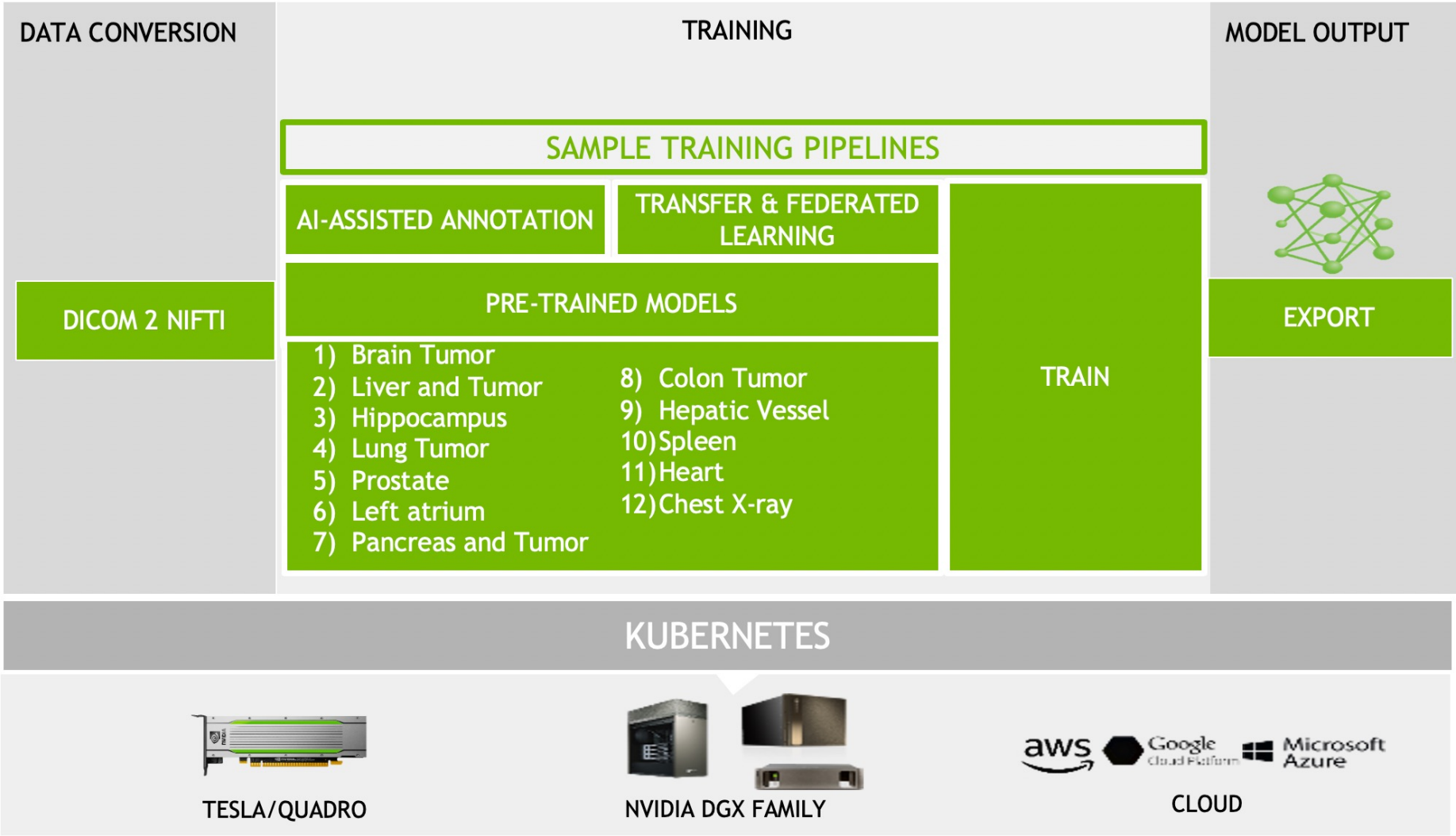
#### 1. Deep Learning Profiler (DLProf)

Deep Learning Profiler is a tool for profiling deep learning models to help data scientists understand and improve performance of their models visually via the DLProf Viewer or by analyzing text reports. We will refer to Deep Learning Profiler simply as DLProf.

DLProf generates reports of summary and detailed analysis of GPU activities based on the whole training program. The format of

# NVIDIA Clara Medical Imaging

## Clara Train SDK v4.0



# *Advanced Topics*

# Search for Optimal Neural Network Architecture

## Oral Presentation in CVPR 2021

**DiNTS: Differentiable Neural Network Topology Search for 3D Medical Image Segmentation**

Yufan He[1] Dong Yang[2] Holger Roth[2] Can Zhao[2] Daguang Xu[2]
[1]Johns Hopkins University [2]NVIDIA

**Abstract**

*Recently, neural architecture search (NAS) has been applied to automatically search high-performance networks for medical image segmentation. The NAS search space usually contains a network topology level (controlling connections among cells with different spatial scales) and a cell level (operations within each cell). Existing methods either require long searching time for large-scale 3D image datasets, or are limited to pre-defined topologies (such as U-shaped or single-path) . In this work, we focus on three important aspects of NAS in 3D medical image segmentation: flexible multi-path network topology, high search efficiency, and budgeted GPU memory usage. A novel differentiable search framework is proposed to support fast gradient-based search within a highly flexible network topology search space. The discretization of the searched optimal continuous model in differentiable scheme may produce a sub-optimal final discrete model (discretization gap). Therefore, we propose a topology loss to alleviate this problem. In addition, the GPU memory usage for the searched 3D model is limited with budget constraints during search. Our **D**ifferentiable **N**etwork **T**opology **S**earch scheme (**DiNTS**) is evaluated on the Medical Segmentation Decathlon (MSD) challenge, which contains ten challeng-*

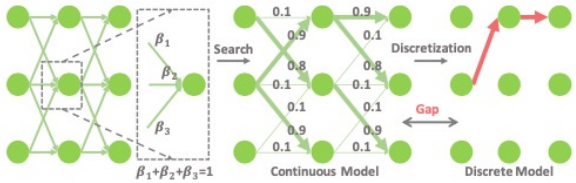



Figure 1. Limitations of existing differentiable topology search formulation. E.g. in Auto-DeepLab [21], each edge in the topology search space is given a probability $\beta$. The probabilities of input edges to a node sum to one, which means only one input edge for each node would be selected. A single-path discrete model (red path) is extracted from the continuous searched model. This can result in a large "discretization gap" between the feature flow of the searched continuous model and the final discrete model.

can vary considerably. This makes the direct application of even a successful network like U-Net [34] to a new task less likely to be optimal.

The neural architecture search (NAS) algorithms [49] have been proposed to automatically discover the optimal architectures within a search space. The NAS search space for segmentation usually contains two levels: network topology level and cell level. The network topology controls the connections among cells and decides the flow

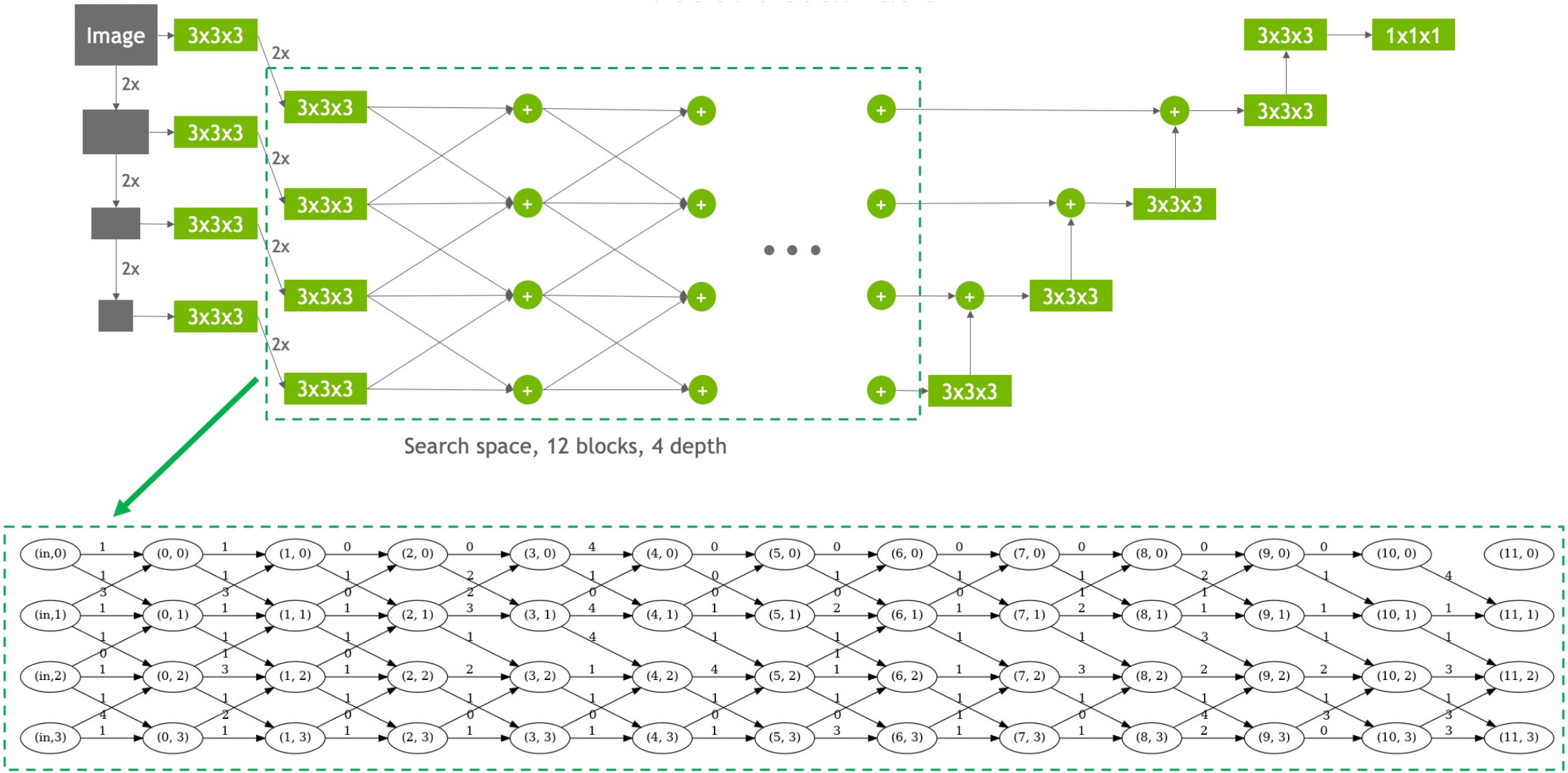
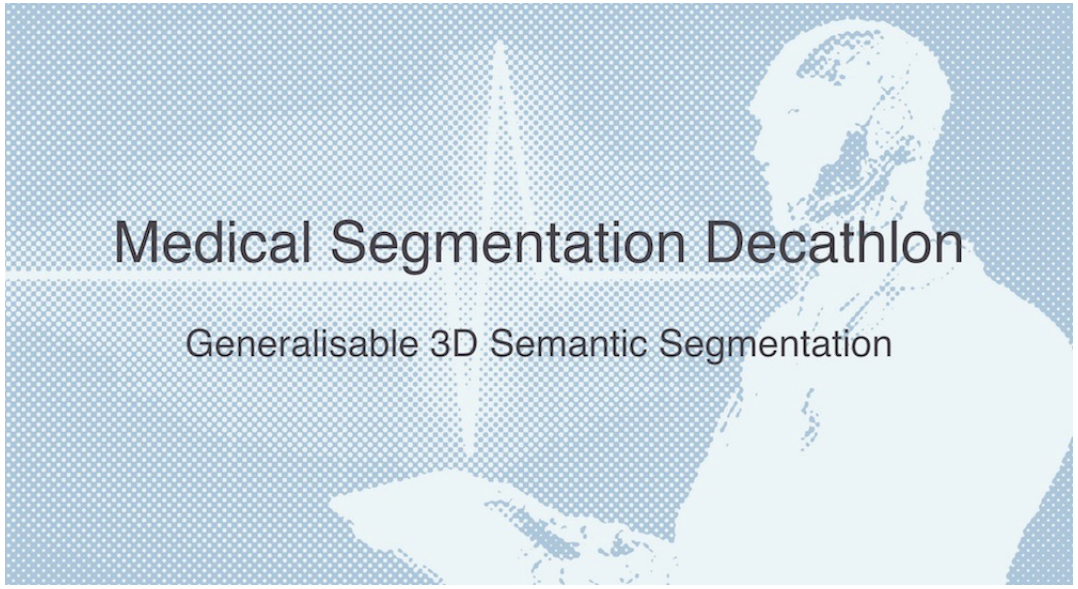


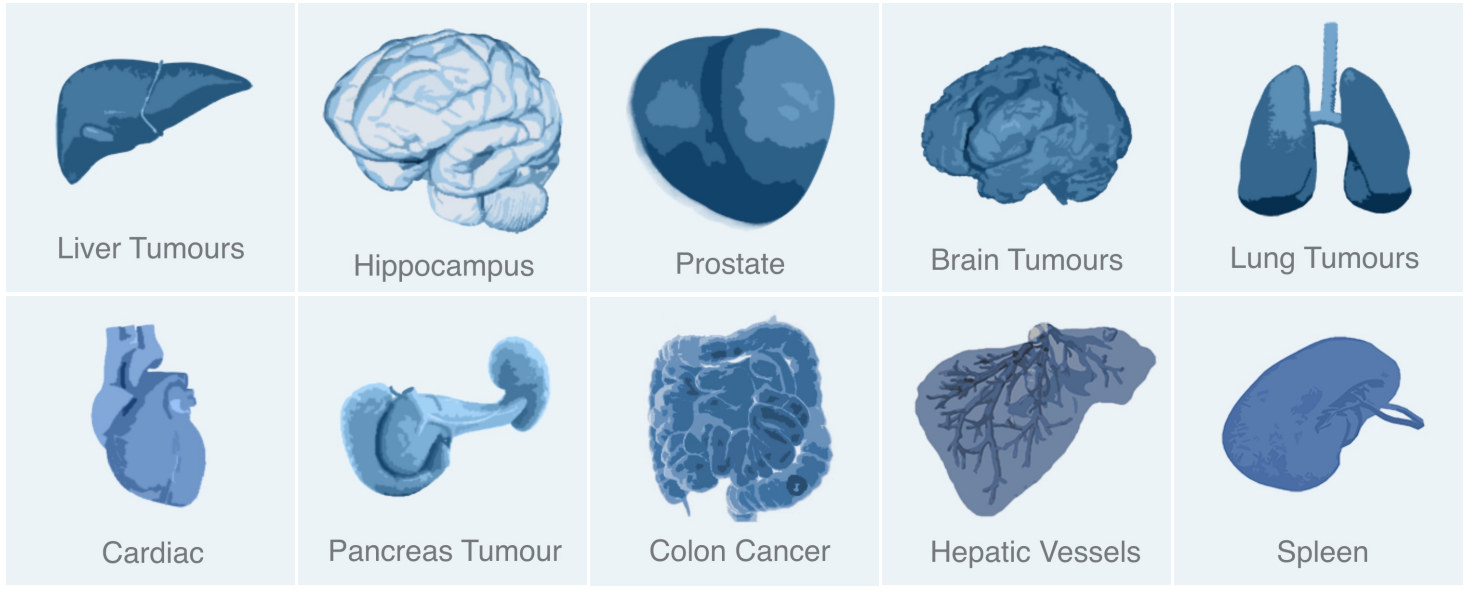
Image
3x3x3
2x
3x3x3
1x1x1
Search space, 12 blocks, 4 depth
(in,0)
(in,1)
(in,2)
(in,3)
(0, 0)
(0, 1)
(0, 2)
(0, 3)
(1, 0)
(1, 1)
(1, 2)
(1, 3)
(2, 0)
(2, 1)
(2, 2)
(2, 3)
(3, 0)
(3, 1)
(3, 2)
(3, 3)
(4, 0)
(4, 1)
(4, 2)
(4, 3)
(5, 0)
(5, 1)
(5, 2)
(5, 3)
(6, 0)
(6, 1)
(6, 2)
(6, 3)
(7, 0)
(7, 1)
(7, 2)
(7, 3)
(8, 0)
(8, 1)
(8, 2)
(8, 3)
(9, 0)
(9, 1)
(9, 2)
(9, 3)
(10, 0)
(10, 1)
(10, 2)
(10, 3)
(11, 0)
(11, 1)
(11, 2)
(11, 3)

# Dataset

## Medical Segmentation Decathlon (MSD)

*Public challenge with 10 different tasks of 3D medical image segmentation*




*Work-in-progress for future release*

*What's more?*

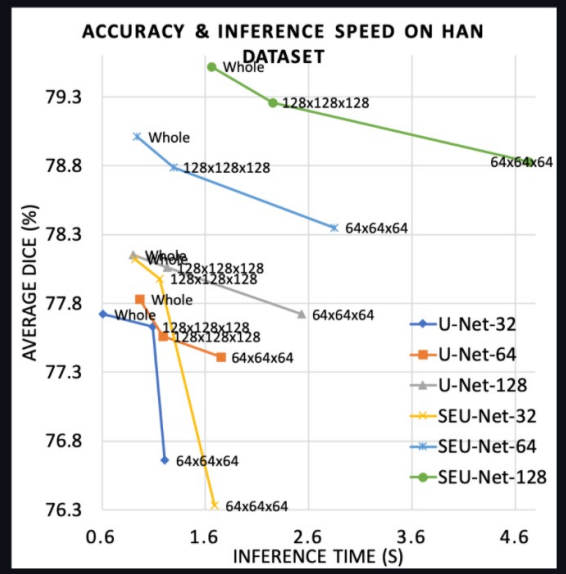
master ▾ research-contributions / lamp-automated-model-parallelism / Go to file Add file ···

wyli Update README.md f7af551 on Oct 1, 2020 History

| | | |
|---|---|---|
| .. | | |
| fig | initial copy from project-MONAI/research | 12 months ago |
| README.md | Update README.md | 12 months ago |
| __init__.py | initial copy from project-MONAI/research | 12 months ago |
| data_utils.py | initial copy from project-MONAI/research | 12 months ago |
| test_unet_pipe.py | initial copy from project-MONAI/research | 12 months ago |
| train.py | initial copy from project-MONAI/research | 12 months ago |
| unet_pipe.py | initial copy from project-MONAI/research | 12 months ago |

README.md

# LAMP: Large Deep Nets with Automated Model Parallelism for Image Segmentation




> If you use this work in your research, please cite the paper.

A reimplementation of the LAMP system originally proposed by:

Wentao Zhu, Can Zhao, Wenqi Li, Holger Roth, Ziyue Xu, and Daguang Xu (2020) "LAMP: Large Deep Nets with Automated Model Parallelism for Image Segmentation." MICCAI 2020 (Early Accept, paper link: https://arxiv.org/abs/2006.12575)

## To run the demo:

master ▾ research-contributions / UNETR / BTCV /

Go to file | Add file ▾ | ···

ahatamiz and wyli add UNETR ... ae00f99 on Jul 14 History

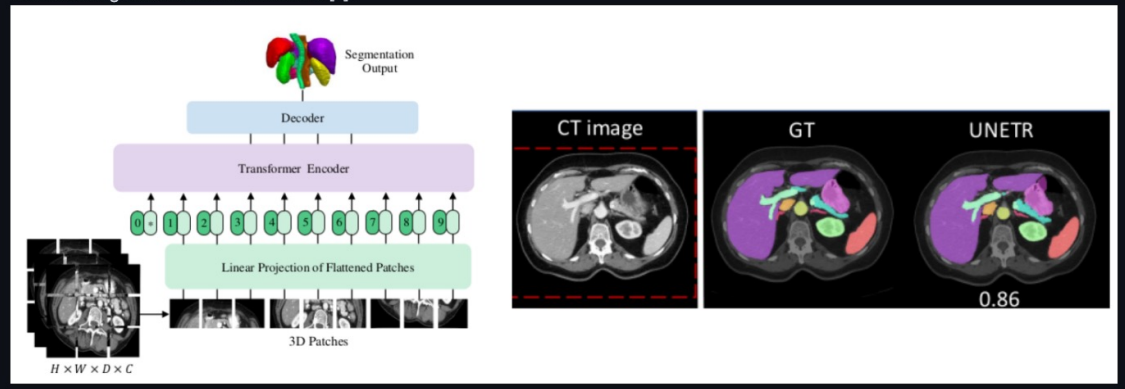
..

| | | |
|---|---|---|
| assets | add UNETR | 2 months ago |
| commands | add UNETR | 2 months ago |
| docs | add UNETR | 2 months ago |
| jsons | add UNETR | 2 months ago |
| networks | add UNETR | 2 months ago |
| optimizers | add UNETR | 2 months ago |
| tests | add UNETR | 2 months ago |
| tutorial | add UNETR | 2 months ago |
| utils | add UNETR | 2 months ago |
| .gitkeep | add UNETR | 2 months ago |
| README.md | add UNETR | 2 months ago |
| __init__.py | add UNETR | 2 months ago |
| __main__.py | add UNETR | 2 months ago |
| requirements.sh | add UNETR | 2 months ago |

README.md

# Model Overview

A transformer-based model for volumetric (3D) multi-organ segmentation task using the BTCV challenge dataset. This model is trained using the UNETR architecture [1].




## Tutorial

A step-by-step tutorial can be found in:

tutorial/unetr_btcv_segmentation_3d.ipynb

# Federated Learning with MONAI using NVFlare (without docker)

The purpose of this tutorial is to show how to run NVFlare with MONAI on a local machine to simulate a FL setting (server and client communicate over localhost). It is based on the tutorial showing how to run FL with MONAI and NVFlare which using a docker container for the server and each client.

## Environment setup

(If needed) install pip and virtualenv (on macOS and Linux):

```
python3 -m pip install --user --upgrade pip
python3 -m pip install --user virtualenv
```

(If needed) make all shell scripts executable using

```
find . -name ".sh" -exec chmod +x {} \;
```

initialize virtual environment and set the current folder (see `projectpath` in `set_env.sh`).

```
source ./virtualenv/set_env.sh
```

install required packages

```
pip install --upgrade pip
pip install -r ${projectpath}/virtualenv/requirements.txt
```

## FL workspace preparation for NVFlare

NVFlare has a "provision" mechanism to automatically generate the fl workspace, see here for details.

In this example, for convenience, we included a pregenerated workspace supporting up to 8 clients which needs to be extracted.

```
unzip ${projectpath}/fl_workspace_pregenerated.zip
```

*Note: (Optional)* If you need to modify the fl workspace (changing the number of max clients, client names, etc.), please follow the instructions here. We included the sample project.yml and authz_config.json files used for generating the 8-client workspace under `${projectpath}/fl_utils/workspace_gen`. After modification, the provisioning tool can be run as: `provision -p project.yml -a authz_config.json`

## Example task - spleen segmentation with MONAI


# Questions