Computer Methods and Programs in Biomedicine 158 (2018) 113–122

Contents lists available at ScienceDirect

Computer Methods and Programs in Biomedicine

journal homepage: www.elsevier.com/locate/cmpb

NiftyNet: a deep-learning platform for medical imaging

Eli Gibson[a,b,1], Wenqi Li[a,1,*], Carole Sudre[b], Lucas Fidon[a], Dzhoshkun I. Shakir[a], Guotai Wang[a], Zach Eaton-Rosen[b], Robert Gray[c,d], Tom Doel[a], Yipeng Hu[b], Tom Whyntie[b], Parashkev Nachev[c,d], Marc Modat[b], Dean C. Barratt[a,b], Sébastien Ourselin[a], M. Jorge Cardoso[b,2], Tom Vercauteren[a,2]

[a] Wellcome / EPSRC Centre for Interventional and Surgical Sciences (WEISS), University College London, UK
[b] Centre for Medical Image Computing (CMIC), Departments of Medical Physics & Biomedical Engineering and Computer Science, University College London, UK
[c] Institute of Neurology, University College London, UK
[d] National Hospital for Neurology and Neurosurgery, London, UK

ARTICLE INFO

Article history:
Received 2 October 2017
Revised 8 January 2018
Accepted 24 January 2018

Keywords:
Medical image analysis
Deep learning
Convolutional neural network
Segmentation
Image regression
Generative adversarial network

ABSTRACT

Background and objectives: Medical image analysis and computer-assisted intervention problems are increasingly being addressed with deep-learning-based solutions. Established deep-learning platforms are flexible but do not provide specific functionality for medical image analysis and adapting them for this domain of application requires substantial implementation effort. Consequently, there has been substantial duplication of effort and incompatible infrastructure developed across many research groups. This work presents the open-source NiftyNet platform for deep learning in medical imaging. The ambition of NiftyNet is to accelerate and simplify the development of these solutions, and to provide a common mechanism for disseminating research outputs for the community to use, adapt and build upon.

Methods: The NiftyNet infrastructure provides a modular deep-learning pipeline for a range of medical imaging applications including segmentation, regression, image generation and representation learning applications. Components of the NiftyNet pipeline including data loading, data augmentation, network architectures, loss functions and evaluation metrics are tailored to, and take advantage of, the idiosyncracies of medical image analysis and computer-assisted intervention. NiftyNet is built on the TensorFlow framework and supports features such as TensorBoard visualization of 2D and 3D images and computational graphs by default.

Results: We present three illustrative medical image analysis applications built using NiftyNet infrastructure: (1) segmentation of multiple abdominal organs from computed tomography; (2) image regression to predict computed tomography attenuation maps from brain magnetic resonance images; and (3) generation of simulated ultrasound images for specified anatomical poses.

Conclusions: The NiftyNet infrastructure enables researchers to rapidly develop and distribute deep learning solutions for segmentation, regression, image generation and representation learning applications, or extend the platform to new applications.

1. Introduction

Computer-aided analysis of medical images plays a critical role at many stages of the clinical workflow from population screening and diagnosis to treatment delivery and monitoring. This role is poised to grow as analysis methods become more accurate and cost effective. In recent years, a key driver of such improvements has been the adoption of deep learning and convolutional neural networks in many medical image analysis and computer-assisted intervention tasks.

Deep learning refers to a deeply nested composition of many simple functions (principally linear combinations such as convolutions, scalar non-linearities and moment normalizations) parame-

* Corresponding author.
E-mail address: wenqi.li@ucl.ac.uk (W. Li).
[1] Wenqi Li and Eli Gibson contributed equally to this work.
[2] M. Jorge Cardoso and Tom Vercauteren contributed equally to this work.

https://doi.org/10.1016/j.cmpb.2018.01.025

terized by variables. The particular composition of functions, called the architecture, defines a parametric function (typically with millions of parameters) that can be optimized to minimize an objective, or 'loss', function, usually using some form of gradient descent.

Although the first use of neural networks for medical image analysis dates back more than twenty years [35], their usage has increased by orders of magnitude in the last five years. Recent reviews [34,51] have highlighted that deep learning has been applied to a wide range of medical image analysis tasks (segmentation, classification, detection, registration, image reconstruction, enhancement, etc.) across a wide range of anatomical sites (brain, heart, lung, abdomen, breast, prostate, musculature, etc.). Although each of these applications have their own specificities, there is substantial overlap in software pipelines implemented by many research groups.

Deep-learning pipelines for medical image analysis comprise many interconnected components. Many of these are common to all deep-learning pipelines:

- separation of data into training, testing and validation sets;
- randomized sampling during training;
- image data loading and sampling;
- data augmentation;
- a network architecture defined as the composition of many simple functions;
- a fast computational framework for optimization and inference;
- metrics for evaluating performance during training and inference.

In medical image analysis, many of these components have domain specific idiosyncrasies, detailed in Section 4. For example, medical images are typically stored in specialized formats that handle large 3D images with anisotropic voxels and encode additional spatial information and/or patient information, requiring different data loading pipelines. Processing large volumetric images has high memory requirements and motivates domain-specific memory-efficient networks or custom data sampling strategies. Images are often acquired in standard anatomical views and can represent physical properties quantitatively, motivating domain-specific data augmentation and model priors. Additionally, the clinical implications of certain errors may warrant custom evaluation metrics. Independent reimplementation of all of this custom infrastructure results in substantial duplication of effort, poses a barrier to dissemination of research tools and inhibits fair comparisons between competing methods.

This work presents the open-source NiftyNet[3] platform to 1) facilitate efficient deep learning research in medical image analysis and computer-assisted intervention; and 2) reduce duplication of effort. The NiftyNet platform comprises an implementation of the common infrastructure and common networks used in medical imaging, a database of pre-trained networks for specific applications and tools to facilitate the adaptation of deep learning research to new clinical applications with a shallow learning curve.

2. Background

The development of common software infrastructure for medical image analysis and computer-assisted intervention has a long history. Early efforts included the development of medical imaging file formats (e.g. ACR-NEMA (1985), Analyze 7.5 (1986), DICOM (1992) MINC (1992), and NIfTI (2001)). Toolsets to solve common challenges such as registration (e.g. NiftyReg [42], ANTs [2] and elastix [31]), segmentation (e.g. NiftySeg [8]), and biomechanical modeling (e.g. [29]) are available for use as part of image analysis pipelines. Pipelines for specific research applications such as FSL [52] for functional MRI analysis and Freesurfer [14,19] for structural neuroimaging have reached widespread use. More general toolkits offering standardized implementations of algorithms (VTK and ITK [44]) and application frameworks (NifTK [12], MITK [43] and 3D Slicer [44]) enable others to build their own pipelines. Common software infrastructure has supported and accelerated medical image analysis and computer-assisted intervention research across hundreds of research groups. However, despite the wide availability of general purpose deep learning software tools, deep learning technology has limited support in current software infrastructure for medical image analysis and computer-assisted intervention.

Software infrastructure for general purpose deep learning is a recent development. Due to the high computational demands of training deep learning models and the complexity of efficiently using modern hardware resources (general purpose graphics processing units and distributed computing, in particular), numerous deep learning libraries and platforms have been developed and widely adopted, including cuDNN [9], TensorFlow [1], Theano [4], Caffe [28], Torch [13], CNTK [50], and MatConvNet [54].

These platforms facilitate the definition of complex deep learning networks as compositions of simple functions, hide the complexities of differentiating the objective function with respect to trainable parameters during training, and execute efficient implementations of performance-critical functions during training and inference. These frameworks have been optimized for performance and flexibility, and using them directly can be challenging, inspiring the development of platforms that simplify the development process for common usage scenarios, such as Keras [10], and TensorLayer [17] for TensorFlow and Lasagne [15] for Theano. However, by avoiding assumptions about the application to remain general, the platforms are unable to provide specific functionality for medical image analysis and adapting them for this domain of application requires substantial implementation effort.

Developed concurrently with the NiftyNet platform, the Deep Learning Toolkit[4] aims to support fast prototyping and reproducibility by implementing deep learning methods and modules for medical image analysis. While still in preliminary development, it appears to focus on deep learning building blocks rather than analysis pipelines. NifTK [12,24] and Slicer3D (via the DeepInfer [38] plugin) provide infrastructure for distribution of trained deep learning pipelines. Although this does not address the substantial infrastructure needed for training deep learning pipelines, integration with existing medical image analysis infrastructure and modular design makes these platforms promising routes for distributing deep-learning pipelines.

3. Typical deep learning pipeline

Deep learning adopts the typical machine learning pipeline consisting of three phases: model selection (picking and fitting a model on training data), model evaluation (measuring the model performance on testing data), and model distribution (sharing the model for use on a wider population). Within these simple phases lies substantial complexity, illustrated in Fig. 1. The most obvious complexity is in implementing the network being studied. Deep neural networks generally use simple functions, but compose them in complex hierarchies; researchers must implement the network being tested, as well as previous networks (often incompletely specified) for comparison. To train, evaluate and distribute these networks, however, requires further infrastructure. Data sets must be correctly partitioned to avoid biassed evaluations, sometimes

[3] Available at http://niftynet.io.

[4] https://dltk.github.io.

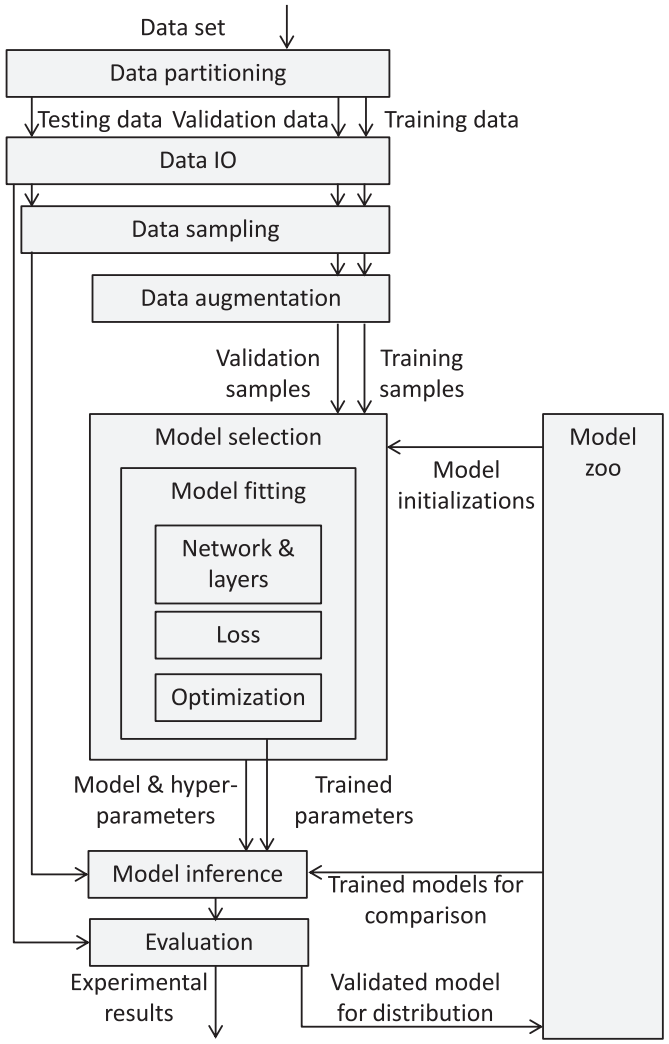

Fig. 1. Data flow implemented in typical deep learning projects. Boxes represent the software infrastructure to be developed and arrows represent the data flow.

considering data correlations (e.g. images acquired at the same hospital may be more similar to each other than to those from other hospitals). The data must be sampled, loaded and passed to the network, in different ways depending on the phase of the pipeline. Algorithms for tuning hyper-parameters within a family of models and optimizing model parameters on the training data are needed. Logging and visualization are needed to debug and dissect models during and after training. In applications with limited data, data sets must be augmented by perturbing the training data in realistic ways to prevent over-fitting. In deep learning, it is common practice to adapt previous network architectures, trained or untrained, in part or in full for similar or different tasks; this requires a community repository (popularly called a *model zoo*) storing models and parameters in an adaptable format. Much of this infrastructure is recreated by each researcher or research group undertaking a deep learning project, and much of it depends on the application domain being addressed.

4. Design considerations for deep learning in medical imaging

Medical image analysis differs from other domains where deep learning is applied due to characteristics of the data itself, and the applications in which they are used. In this section, we present the domain-specific requirements driving the design of NiftyNet.

4.1. Data availability

Acquiring, annotating and distributing medical image data sets have higher costs than in many computer vision tasks. For many medical imaging modalities, generating an image is costly. Annotating images for many applications requires high levels of expertise from clinicians with limited time. Additionally, due to privacy concerns, sharing data sets between institutions, let alone internationally, is logistically and legally challenging. Although recent tools such as DeepIGeoS [56] for semi-automated annotation and GIFT-Cloud [16] for data sharing are beginning to reduce these barriers, typical data sets remain small. Using smaller data sets increases the importance of data augmentation, regularization, and cross-validation to prevent over-fitting. The additional cost of data set annotation also places a greater emphasis on semi- and unsupervised learning.

4.2. Data dimensionality and size

Data dimensionality encountered in medical image analysis and computer-assisted intervention typically ranges from 2D to 5D. Many medical images, including MRI, CT, PET and SPECT, capture volumetric images. Longitudinal imaging (multiple images taken over time) is typical in interventional settings as well as clinically useful for measuring organ function (e.g. blood ejection fraction in cardiac imaging) and disease progression (e.g. cortical thinning in neurodegenerative diseases).

At the same time, capturing high-resolution data in multiple dimensions is often necessary to detect small but clinically important anatomy and pathology. The combination of these factors results in large data sizes for each sample, which impact computational and memory costs. Deep learning in medical imaging uses various strategies to account for this challenge. Many networks are designed to use partial images: 2D slices sampled along one axis from 3D images [57], 3D subvolumes [33], anisotropic convolution [55], or combinations of subvolumes along multiple axes [48]. Other networks use multi-scale representations allowing deeper and wider networks on lower-resolution representations [30,40]. A third approach uses dense networks to reuse feature representations multiple times in the network [23]. Smaller batch sizes can reduce the memory cost, but rely on different weight normalization functions such as batch renormalization [27], weight normalization [49] or layer normalization [3].

4.3. Data formatting

Data sets in medical imaging are typically stored in different formats than in many computer vision tasks. To support the higher-dimensional medical image data, specialized formats have been adopted (e.g. DICOM, NIfTI, Analyze). These formats frequently also store metadata that is critical to image interpretation, including spatial information (anatomical orientation and voxel anisotropy), patient information (demographics and identifiers), and acquisition information (modality types and scanner parameters). These medical imaging specific data formats are typically not supported by existing deep learning frameworks, requiring custom infrastructure for loading images.

4.4. Data properties

The characteristic properties of medical image content pose opportunities and challenges. Medical images are obtained under controlled conditions, allowing more predictable data distributions. In many modalities, images are calibrated such that spatial relationships and image intensities map directly to physical quan-

tities and are inherently normalized across subjects. For a given clinical workflow, image content is typically consistent, potentially enabling the characterization of plausible intensity and spatial variation for data augmentation. However, some clinical applications introduce additional challenges. Because small image features can have large clinical importance, and because some pathology is very rare but life-threatening, medical image analysis must deal with large class imbalances, motivating special loss functions [18,40,53]. Furthermore, different types of error may have very different clinical impacts, motivating specialized loss functions and evaluation metrics (e.g. spatially weighted segmentation metrics). Applications in computer-assisted intervention where analysis results are used in real time (e.g. [21,24]) have additional constraints on analysis latency.

5. NiftyNet: a platform for deep learning in medical imaging

The NiftyNet platform aims to augment the current deep learning infrastructure to address the ideosyncracies of medical imaging described in Section 4, and lower the barrier to adopting this technology in medical imaging applications. NiftyNet is built using the TensorFlow library, which provides the tools for defining computational pipelines and executing them efficiently on hardware resources, but does not provide any specific functionality for processing medical images, or high-level interfaces for common medical image analysis tasks. NiftyNet provides a high-level deep learning pipeline with components optimized for medical imaging applications (data loading, sampling and augmentation, networks, loss functions, evaluations, and a model zoo) and specific interfaces for medical image segmentation, classification, regression, image generation and representation learning applications.

5.1. Design goals

The design of NiftyNet follows several core principles which support a set of key requirements:

- support a wide variety of application types in medical image analysis and computer-assisted intervention;
- enable research in one aspect of the deep learning pipeline without the need for recreating the other parts;
- be simple to use for common use cases, but flexible enough for complex use cases;
- support built-in TensorFlow features (parallel processing, visualization) by default;
- support best practices (data augmentation, data set separation) by default;
- support model distribution and adaptation.

5.2. System overview

The NiftyNet platform comprises several modular components. The TensorFlow framework defines the interface for and executes the high performance computations used in deep learning. The NiftyNet `ApplicationDriver` defines the common structure across all applications, and is responsible for instantiating the data analysis pipeline and distributing the computation across the available computational resources. The NiftyNet `Application` classes encapsulate standard analysis pipelines for different medical image analysis applications, by connecting four components: a `Reader` to load data from files, a `Sampler` to generate appropriate samples for processing, a `Network` to process the inputs, and an output handler (comprising the `Loss` and `Optimizer` during training and an `Aggregator` during inference and evaluation). The `Sampler` includes sub-components for data augmentation. The

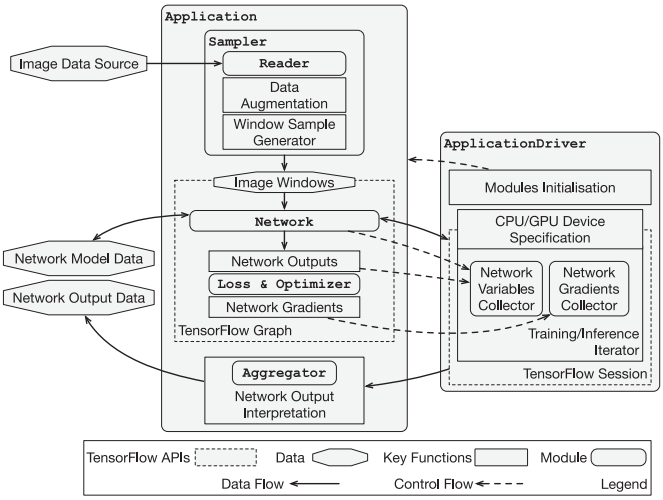

Fig. 2. A brief overview of NiftyNet components.

`Network` includes sub-components representing individual network blocks or larger conceptual units. These components are briefly depicted in Fig. 2 and detailed in the following sections.

As a concrete illustration, one instantiation of the `SegmentationApplication` could use the following modules. During training, it could use a `UniformSampler` to generate small image patches and corresponding labels; a `vnet Network` would process batches of images to generate segmentations; a `Dice LossFunction` would compute the loss used for backpropagation using the `Adam Optimizer`. During inference, it could use a `GridSampler` to generate a set of non-overlapping patches to cover the image to segment, the same network to generate corresponding segmentations, and a `GridSamplesAggregator` to aggregate the patches into a final segmentation.

5.3. Component details: TensorFlow framework

The TensorFlow framework defines the interface for and executes the high performance computations used in deep learning. Briefly, TensorFlow provides a Python application programming interface to construct an abstract computation graph comprising composable operations with support for automatic differentiation. The choice of the TensorFlow framework over the many deep learning frameworks described above reflects both engineering concerns – including cross-platform support, multi-GPU support, built-in visualization tools, installation without compilation, and semantic versioning – as well as pragmatic concerns, such as its larger number of users and support from industry.

5.4. Component details: ApplicationDriver class

The NiftyNet `ApplicationDriver` defines the common structure for all NiftyNet pipelines. It is responsible for instantiating the data and `Application` objects and distributing the workload across and recombining results from the computational resources (potentially including multiple CPUs and GPUs). It is also responsible for handling variable initialization, variable saving and restoring, and logging. Implemented as a template design pattern [20], the `ApplicationDriver` delegates application-specific functionality to separate `Application` classes.

The `ApplicationDriver` can be configured from the command line or programmatically using a human-readable configuration file. This file contains the data set definitions and all the settings that deviate from the defaults. When the `ApplicationDriver` saves its progress, the full configuration

(including default parameters) is also saved so that the analysis pipeline can be recreated to continue training or carry out inference internally or with a distributed model.

5.5. Component details: application class

Medical image analysis encompasses a wide range of tasks for different parts of the pre-clinical and clinical workflow: segmentation, classification, detection, registration, reconstruction, enhancement, model representation and generation. Different applications use different types of inputs and outputs, different networks, and different evaluation metrics; however, there is common structure and functionality among these applications supported by NiftyNet. NiftyNet currently supports

- image segmentation,
- image regression,
- image model representation (via auto-encoder applications), and
- image generation (via auto-encoder and generative adversarial networks (GANs)),

and it is designed in a modular way to support the addition of new application types, by encapsulating typical application workflows in `Application` classes.

The `Application` class defines the required data interface for the `Network` and `Loss`, facilitates the instantiation of appropriate `Sampler` and output handler objects, connects them as needed for the application, and specifies the training regimen. For example, the `SegmentationApplication` specifies that networks accept images (or patches thereof) and generate corresponding labels, that losses accept generated and reference segmentations and an optional weight map, and that the optimizer trains all trainable variables in each iteration. In contrast, the `GANApplication` specifies that networks accept a noise source, samples of real data and an optional conditioning image, losses accept logits denoting if a sample is real or generated, and the optimizer alternates between training the discriminator sub-network and the generator sub-network.

5.6. Component details: networks and layers

The complex composition of simple functions that comprise a deep learning architecture is simplified in typical networks by the repeated reuse of conceptual blocks. In NiftyNet, these conceptual blocks are represented by encapsulated `Layer` classes, or inline using TensorFlow's scoping system. Composite layers, and even entire networks, can be constructed as simple compositions of NiftyNet layers and TensorFlow operations. This supports the reuse of existing networks by clearly demarcating conceptual blocks of code that can be reused and assigning names to corresponding sets of variables that can be reused in other networks (detailed in Section 5.11). This also enables automatic support for visualization of the network graph as a hierarchy at different levels of detail using the TensorBoard visualizer [37] as shown in Fig. 3. Following the model used in Sonnet [46], `Layer` objects define a scope upon instantiation, which can be reused repeatedly to allow complex weight-sharing without breaking encapsulation.

5.7. Component details: data loading

The `Reader` class is responsible for loading corresponding image files from medical file formats for a specified data set, and applying image-wide preprocessing. For simple use cases, NiftyNet can automatically identify corresponding images in a data set by searching a specified file path and matching user-specified patterns in file names, but it also allows explicitly tabulated comma-separated value files for more complex data set structures (e.g.

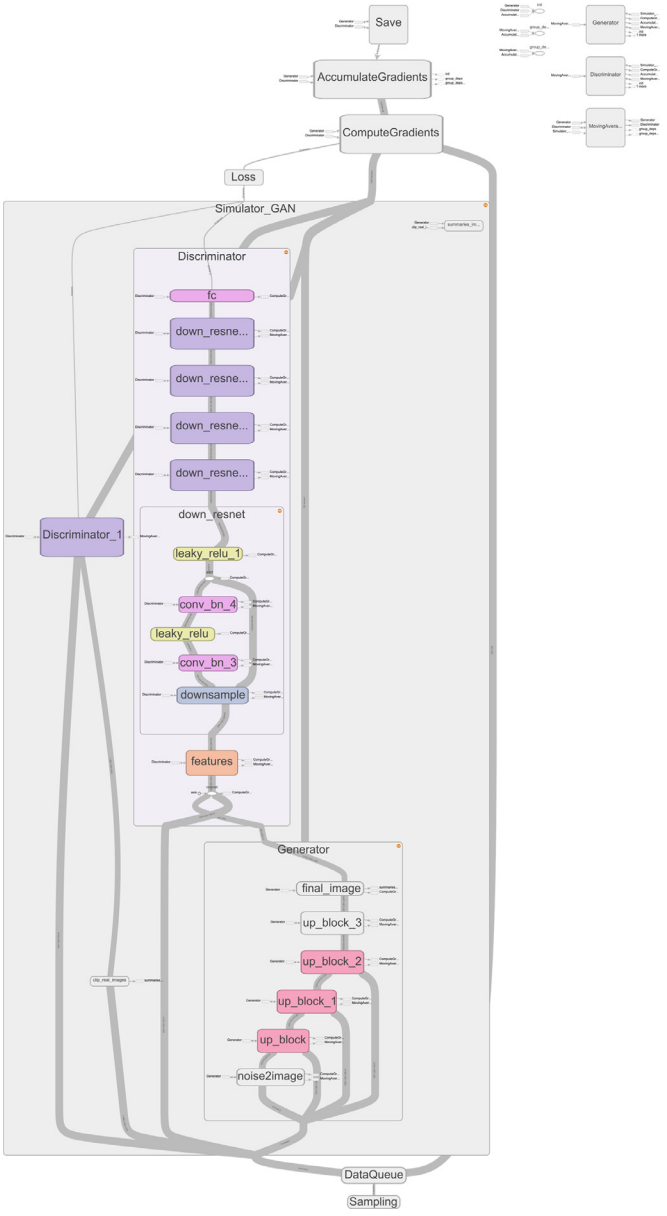

Fig. 3. TensorBoard visualization of a NiftyNet generative adversarial network. TensorBoard interactively shows the composition of conceptual blocks (rounded rectangles) and their interconnections (grey lines) and color-codes similar blocks. Above, the generator and discriminator blocks and one of the discriminator's residual blocks are expanded. Font and block sizes were edited for readability.

cross-validation studies). Input and output of medical file formats are already supported in multiple existing Python libraries, although each library supports different sets of formats. To facilitate a wide range of formats, NiftyNet uses `nibabel` [6] as a core dependency but can fall back on other libraries (e.g. SimpleITK [36] if they are installed and a file format is not supported by `nibabel`. A pipeline of image-wide preprocessing functions, described in Section 5.9, is applied to each image before samples are taken.

5.8. Component details: samplers and output handlers

To handle the breadth of applications in medical image analysis and computer-assisted intervention, NiftyNet provides flexibility in mapping from an input data set into packets of data to be processed and from the processed data into useful outputs. The

former is encapsulated in `Sampler` classes, and the latter is encapsulated in output handlers. Because the sampling and output handling are tightly coupled and depend on the action being performed (i.e. training, inference or evaluation), the instantiation of matching `Sampler` objects and output handlers is delegated to the `Application` class.

`Sampler` objects generate a sequence of packets of corresponding data for processing. Each packet contains all the data for one independent computation (e.g. one step of gradient descent during training), including images, labels, classifications, noise samples or other data needed for processing. During training, samples are taken randomly from the training data, while during inference and evaluation the samples are taken systematically to process the whole data set. To feed these samples to TensorFlow, NiftyNet automatically takes advantage of TensorFlow's data queue support: data can be loaded and sampled in multiple CPU threads, combined into mini-batches and consumed by one or more GPUs. NiftyNet includes `Sampler` classes for sampling image patches (uniformly or based on specified criteria), sampling whole images rescaled to a fixed size and sampling noise; and it supports composing multiple `Sampler` objects for more complex inputs.

Output handlers take different forms during training and inference. During training, the output handler takes the network output, computes a loss and the gradient of the loss with respect to the trainable variables, and uses an `Optimizer` to iteratively train the model. During inference, the output handler generates useful outputs by aggregating one or more network outputs and performing any necessary postprocessing (e.g. resizing the outputs to the original image size). NiftyNet currently supports `Aggregator` objects for combining image patches, resizing images, and computing evaluation metrics.

5.9. Component details: data normalization and augmentation

Data normalization and augmentation are two approaches to compensating for small training data sets in medical image analysis, wherein the training data set is too sparse to represent the variability in the distribution of images. Data normalization reduces the variability in the data set by transforming inputs to have specified invariant properties, such as fixed intensity histograms or moments (mean and variance). Data augmentation artificially increases the variability of the training data set by introducing random perturbations during training, for example applying random spatial transformations or adding random image noise. In NiftyNet, data augmentation and normalization are implemented as `Layer` classes applied in the `Sampler`, as plausible data transformations will vary between applications. Some of these layers, such as histogram normalization, are data dependent; these layers compute parameters over the data set before training begins. NiftyNet currently supports mean, variance and histogram intensity data normalization, and flip, rotation and scaling spatial data augmentation.

5.10. Component details: data evaluation

Summarizing and comparing the performance of image analysis pipelines typically rely on standardized descriptive metrics and error metrics as surrogates for performance. Because individual metrics are sensitive to different aspects of performance, multiple metrics are reported together. Reference implementations of these metrics reduce the burden of implementation and prevent implementation inconsistencies. NiftyNet currently supports the calculation of descriptive and error metrics for segmentation. Descriptive statistics include spatial metrics (e.g. volume, surface/volume ratio, compactness) and intensity metrics (e.g. mean, quartiles, skewness of intensity). Error metrics, computed with respect to a reference segmentation, include overlap metrics (e.g. Dice and Jaccard scores; voxel-wise sensitivity, specificity and accuracy), boundary distances (e.g. mean absolute distance and Hausdorff distances) and region-wise errors (e.g. detection rate; region-wise sensitivity, specificity and accuracy).

5.11. Component details: model zoo for network reusability

To support the reuse of network architectures and trained models, many deep learning platforms host a database of existing trained and untrained networks in a standardized format, called a model zoo. Trained networks can be used directly (as part of a workflow or for performance comparisons), fine-tuned for different data distributions (e.g. a different hospital's images), or used to initialize networks for other applications (i.e. transfer learning). Untrained networks or conceptual blocks can be used within new networks. NiftyNet provides several mechanisms to support the distribution and reuse of networks and conceptual blocks.

Trained NiftyNet networks can be restored directly using configuration options. Trained networks developed outside of NiftyNet can be adapted to NiftyNet by encapsulating the network within a `Network` class derived from `TrainableLayer`. Externally trained weights can be loaded within NiftyNet using a `restore_initializer`, adapted from Sonnet [46], for the complete network or individual conceptual blocks. `restore_initializer` initializes the network weights with those stored in a specified checkpoint, and supports `variable_scope` renaming for checkpoints with incompatible scope names. Smaller conceptual blocks, encapsulated in `Layer` classes, can be reused in the same way. Trained networks incorporating previous networks are saved in a self-contained form to minimize dependencies.

The NiftyNet model zoo contains both untrained networks (e.g. `unet` [11] and `vnet` [40] for segmentation), as well as trained networks for some tasks (e.g. `dense_vnet` [22] for multi-organ abdominal CT segmentation, `wnet` [55] for brain tumor segmentation and `simulator_gan` [26] for generating ultrasound images). Model zoo entries should follow a standard format comprising:

- Python source code defining any components not included in NiftyNet (e.g. external `Network` classes, `Loss` functions);
- an example configuration file defining the default settings and the data ordering;
- documentation describing the network and assumptions on the input data (e.g. dimensionality, shape constraints, intensity statistic assumptions).

For trained networks, it should also include:

- a Tensorflow checkpoint containing the trained weights;
- documentation describing the data used to train the network and on which the trained network is expected to perform adequately.

5.12. Platform processes

In addition to the implementation of common functionality, NiftyNet development has adopted good software development processes to support the ease-of-use, robustness and longevity of the platform as well as the creation of a vibrant community. The platform supports easy installation via the `pip` installation tool[5] (i.e. `pip install niftynet`) and provides analysis pipelines that can be run as part of the command line interface. Examples demonstrating the platform in multiple use cases are included to

[5] https://pip.pypa.io.

Table 1
Median segmentation metrics for 8 organs aggregated over the 9-fold cross-validation.

	Dice score	Relative volume difference	Mean absolute distance (voxels)	95th percentile Hausdorff distance (voxels)
Spleen	0.94	0.03	1.07	2.00
L. Kidney	0.93	0.04	1.06	3.00
Gallbladder	0.79	0.17	1.55	4.41
Esophagus	0.68	0.57	2.05	6.00
Liver	0.95	0.02	1.42	4.12
Stomach	0.87	0.09	2.06	8.88
Pancreas	0.75	0.19	1.93	7.62
Duodenum	0.62	0.24	3.05	12.47

reduce the learning curve. The NiftyNet repository uses continuous integration incorporating system and unit tests for regression testing. To mitigate issues due to library version compatibility, NiftyNet releases will follow two policies: (1) the range of compatible versions of NiftyNet dependencies will be encoded in a `requirements.txt` file in the code repository enabling automatic installation of compatible libraries for any NiftyNet version, and (2) NiftyNet versions will follow the semantic versioning 2.0 standard [45] to ensure clear communication regarding backwards compatibility.

6. Results: illustrative applications

6.1. Abdominal organ segmentation

Segmentations of anatomy and pathology on medical images can support image-guided interventional workflows by enabling the visualization of hidden anatomy and pathology during surgical navigation. Here we present an example, based on a simplified version of [22], that illustrates the use of NiftyNet to train a *Dense V-network* to segment organs on abdominal CT that are important to pancreatobiliary interventions: the gastrointestinal tract (esophagus, stomach and duodenum), the pancreas, and anatomical landmark organs (liver, left kidney, spleen and stomach).

The data used to train the network comprised 90 abdominal CT with manual segmentations from two publicly available data sets [32,47], with additional manual segmentations performed at our centre.

The network was trained and evaluated in a 9-fold cross-validation, using the network implementation available in NiftyNet. Briefly, the network, available as `dense_vnet` in NiftyNet, uses a V-shaped structure (with downsampling, upsampling and skip connections) where each downsampling stage is a dense feature stack (i.e. a sequence of convolution blocks where the inputs are concatenated features from all preceding convolution blocks), upsampling is bilinear upsampling and skip connections are convolutions. The loss is a modified Dice loss (with additional hinge losses to mitigate class imbalance) implemented external to NiftyNet and included via a reference in the configuration file. The network was trained for 3000 iterations on whole images (using the `ResizeSampler`) with random affine spatial augmentations.

Segmentation metrics, computed using NiftyNet's `evaluation` action, and aggregated over all folds, are given in Table 1. The segmentation with Dice scores closest to the median is shown in Fig. 4.

Because this network was initially developed prior to NiftyNet and later re-developed for inclusion in NiftyNet, a comparison of the two implementations illustrates the relative advantages of developing with NiftyNet.

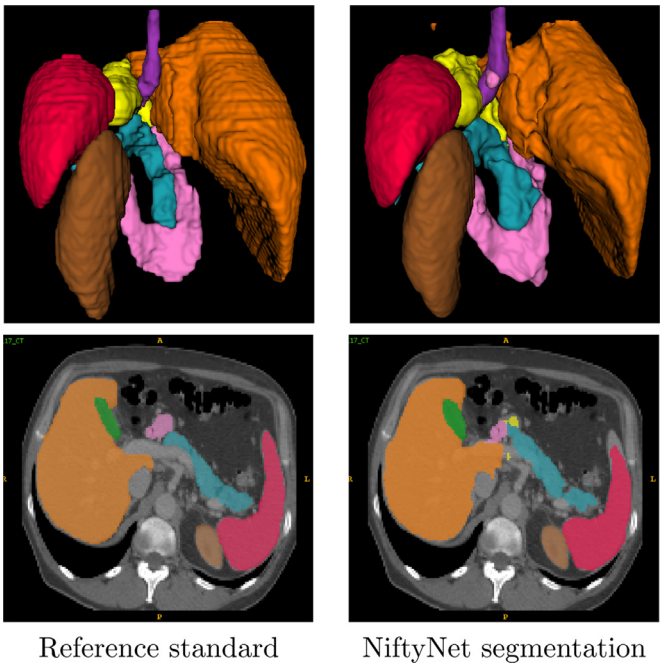

Fig. 4. Reference standard (left) and NiftyNet (right) multi-organ abdominal CT segmentation for the subject with Dice scores closest to the median. Each segmentation is shown with a surface rendering view from the posterior direction and with organ labels overlaid on a transverse CT slice.

The pre-NiftyNet implementation used TensorFlow directly for deep learning and used custom MATLAB code and third-party MATLAB libraries for converting data from medical image formats, pre-/post-processing and evaluating the inferred segmentations. In addition to python code implementing the novel aspects of the work (e.g. a new memory-efficient dropout implementation and a new network architecture), additional infrastructure was developed to load data, separate the data for cross-validation, sample training and validation data, resample images for data augmentation, organise model snapshots, log intermediate losses on training and validation sets, coordinate each experiment, and compute inferred segmentations on the test set. The pre-NiftyNet implementation was not conducive to distributing the code or the trained network, and lacked visualizations for monitoring segmentation performance during training.

In contrast, the NiftyNet implementation was entirely Python-based and required implementations of custom network, data augmentation and loss functions specific to the new architecture, including four conceptual blocks to improve code readability. The network was trained using images in their original NIfTI medical image format and the resulting trained model was publicly deployed in the NiftyNet model zoo. Furthermore, now that the DenseVNet architecture is incorporated into NiftyNet, the network and its conceptual blocks can be used in new segmentation problems with no code development using the command line interface.

6.2. Image regression

Image regression, more specifically, the ability to predict the content of an image given a different imaging modality of the same object, is of paramount importance in real-world clinical workflows. Image reconstruction and quantitative image analysis algorithms commonly require a minimal set of inputs that are often not be available for every patient due to the presence of imaging artefacts, limitations in patient workflow (e.g. long acquisition

Table 2
The Mean Absolute Error (MAE) and the Mean Error (ME) between the ground truth and the pseudoCT in Hounsfield units, comparing the NiftyNet method with pCT [7] and the UTE-based method of the Siemens Biograph mMR.

		NiftyNet	pCT	UTE
MAE	Average	88	121	203
	S.D	7.5	17	24
ME	Average	9.1	−7.3	−132
	S.D.	12	23	34

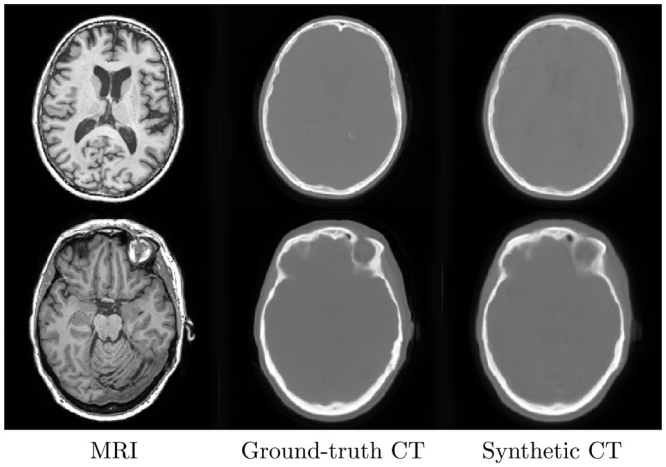

Fig. 5. The input T1 MRI image (left), the ground truth CT (centre) and the NiftyNet regression output (right).

time), image harmonization, or due to ionising radiation exposure minimization.

An example application of image regression is the process of generating synthetic CT images from MRI data to enable the attenuation correction of PET-MRI images [7]. This regression problem has been historically solved with patch-based or multi-atlas propagation methods, a class of models that are very robust but computationally complex and dependent on image registration. The same process can now be solved using the deep learning architectures similar to the ones used in image segmentation.

As a demonstration of this application, a neural network was trained and evaluated in a 5-fold cross-validation setup using the `net_regress` application in NiftyNet. Briefly, the network, available as `highresnet` in NiftyNet, uses a stack of residual dilated convolutions with increasingly large dilation factors [33]. The root mean square error was used as the loss function and implemented as part of NiftyNet as `rmse`. The network was trained for 15,000 iterations on patches of size $80 \times 80 \times 80$, and using the `iSampler` [5] for patch selection with random affine spatial augmentations.

Regression metrics, computed using NiftyNet's 'evaluation' action, and aggregated over all folds, are given in Table 2. The 25th and 75th percentile example result with regards to MAE is shown in Fig. 5.

6.3. Ultrasound simulation using generative adversarial networks

Generating plausible images with specified image content can support training for radiological or image-guided interventional tasks. Conditional GANs have shown promise for generating plausible photographic images [41]. Recent work on spatially-conditioned GANs [26] suggests that conditional GANs could enable software-based simulation in place of costly physical ultrasound phantoms used for training. Here we present an example illustrating a pre-trained ultrasound simulation network that was ported to NiftyNet for inclusion in the NiftyNet model zoo.

The network was originally trained outside of the NiftyNet platform as described in [26]. Briefly, a conditional GAN network was trained to generate ultrasound images of specified views of a fetal phantom using 26,000 frames of optically tracked ultrasound. An image can be sampled from the generative model based on a conditioning image (denoting the pixel coordinates in 3D space) and a model parameter (sampled from a 100-D Gaussian distribution).

The network was ported to NiftyNet for inclusion in the model zoo. The network weights were transferred to the NiftyNet network using NiftyNet's `restore_initializer`, adapted from Sonnet [46], which enables trained variables to be loaded from networks with different architectures or naming schemes.

The network was evaluated multiple times using the `linear_interpolation` inference in NiftyNet, wherein samples are taken from the generative model based on one conditioning image and a sequence of model parameters evenly interpolated between two random samples. Two illustrative results are shown in Fig. 6. The first shows the same anatomy, but a smooth transition between different levels of ultrasound shadowing artifacts. The second shows a sharp transition in the interpolation, suggesting the presence of mode collapse, a common issue in GANs [25].

7. Discussion

7.1. Lessons learned

NiftyNet development was guided by several core principles that impacted the implementation. Maximizing simplicity for simple use cases motivated many implementation choices. We envisioned three categories of users: novice users who are comfortable with running applications, but not with writing new Python code, intermediate users who are comfortable with writing some code, but not with modifying the NiftyNet libraries, and advanced users who are comfortable with modifying the libraries. Support for `pip` installation simplifies NiftyNet for novice and intermediate users. In this context, enabling experimental manipulation of individual pipeline components for intermediate users, and downloadable model zoo entries with modified components for novice users required a modular approach with plugin support for externally defined components. Accordingly, plugins for networks, loss functions and even application logic can be specified by Python `import` paths directly in configuration files without modifying the NiftyNet library. Intermediate users can customize pipeline components by writing classes or functions in Python, and can embed them into model zoo entries for distribution.

Although initially motivated by simplifying variable sharing within networks, NiftyNet's named conceptual blocks also simplified the adaptation of weights from pre-trained models and the TensorBoard-based hierarchical visualization of the computation graphs. The scope of each conceptual blocks maps to a meaningful subgraph of the computation graph and all associated variables, meaning that all weights for a conceptual block can be loaded into a new model with a single scope reference. Furthermore, because these conceptual blocks are constructed hierarchically through the composition of `Layer` objects and scopes, they naturally encode a hierarchical structure for TensorBoard visualization

Supporting machine learning for a wide variety of application types motivated the separation of the `ApplicationDriver` logic that is common to all applications from the `Application` logic that varies between applications. This facilitated the rapid development of new application types. The early inclusion of both image segmentation/regression (mapping from images to images) and image generation (mapping from parameters to images)

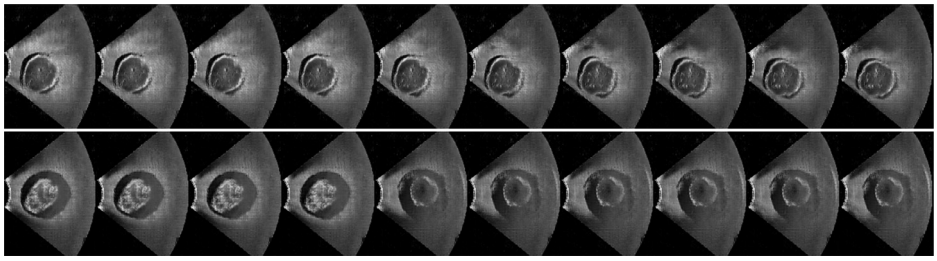

Fig. 6. Interpolated images from the generative model space based on linearly interpolated model parameters. The top row shows a smooth variation between different amounts of ultrasound shadow artefacts. The bottom row shows a sharp transition suggesting the presence of mode collapse in the generative model.

motivated a flexible specification for the number, type and semantic meaning of inputs and outputs, encapsulated in the `Sampler` and `Aggregator` components.

7.2. Platform availability

The NiftyNet platform is available from http://niftynet.io/. The source code can be accessed from the Git repository[6] or installed as a Python library using `pip install niftynet`. NiftyNet is licensed under an open-source Apache 2.0 license[7]. The NiftyNet Consortium welcomes contributions to the platform and seeks inclusion of new community members to the consortium.

7.3. Future direction

The active NiftyNet development roadmap is focused on three key areas: new application types, a larger model zoo and more advanced experimental design. NiftyNet currently supports image segmentation, regression, generation and representation learning applications. Future applications under development include image classification, registration, and enhancement (e.g. super-resolution) as well as pathology detection. The current NiftyNet model zoo contains a small number of models as proof of concept; expanding the model zoo to include state-of-the-art models for common tasks and public challenges (e.g. brain tumor segmentation (BRaTS) [39,55]); and models trained on large data sets for transfer learning will be critical to accelerating research with NiftyNet. Finally, NiftyNet currently supports a simplified machine learning pipeline that trains a single network, but relies on users for data partitioning and model selection (e.g. hyper-parameter tuning). Infrastructure to facilitate more complex experiments, such as built-in support for cross-validation and standardized hyper-parameter tuning will, in the future, reduce the implementation burden on users.

8. Summary of contributions and conclusions

This work presents the open-source NiftyNet platform for deep learning in medical imaging. Our modular implementation of the typical medical imaging machine learning pipeline allows researchers to focus implementation effort on their specific innovations, while leveraging the work of others for the remaining pipeline. The NiftyNet platform provides implementations for data loading, data augmentation, network architectures, loss functions and evaluation metrics that are tailored for the idiosyncracies of medical image analysis and computer-assisted intervention. This infrastructure enables researchers to rapidly develop deep learning solutions for segmentation, regression, image generation and representation learning applications, or extend the platform to new applications.

[6] https://github.com/NifTK/NiftyNet.

[7] https://www.apache.org/licenses/LICENSE-2.0.

Conflict of interest

None.

Acknowledgments

The authors would like to acknowledge all of the contributors to the NiftyNet platform. This work was supported by the Wellcome/EPSRC [203145Z/16/Z, WT101957, NS/A000027/1]; Wellcome [106882/Z/15/Z, WT103709]; the Department of Health and Wellcome Trust [HICF-T4-275, WT 97914]; EPSRC [EP/M020533/1, EP/K503745/1, EP/L016478/1]; the National Institute for Health Research University College London Hospitals Biomedical Research Centre (NIHR BRC UCLH/UCL High Impact Initiative); Cancer Research UK (CRUK) [C28070/A19985]; the Royal Society [RG160569]; a UCL Overseas Research Scholarship, and a UCL Graduate Research Scholarship. The authors would like to acknowledge that the work presented here made use of Emerald, a GPU-accelerated High Performance Computer, made available by the Science & Engineering South Consortium operated in partnership with the STFC Rutherford-Appleton Laboratory; and hardware donated by NVIDIA.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: large-scale machine learning on heterogeneous distributed systems, White Paper, 2016. arXiv: 1603.04467v2.

[2] B.B. Avants, N.J. Tustison, G. Song, P.A. Cook, A. Klein, J.C. Gee, A reproducible evaluation of ANTs similarity metric performance in brain image registration, Neuroimage 54 (3) (2011) 2033–2044.

[3] J.L. Ba, J.R. Kiros, G.E. Hinton, Layer normalization (2016). arXiv:1607.06450v1.

[4] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I.J. Goodfellow, A. Bergeron, N. Bouchard, Y. Bengio, Theano: new features and speed improvements, in: Proceedings of the Workshop on Deep Learning and Unsupervised Feature Learning NIPS, 2012.

[5] L. Berger, E. Hyde, M.J. Cardoso, S. Ourselin, An adaptive sampling scheme to efficiently train fully convolutional networks for semantic segmentation, (2017). ArXiv e-prints arXiv:1709.02764.

[6] M. Brett, M. Hanke, B. Cipollini, M.-A. Côté, C. Markiewicz, S. Gerhard, E. Larson, Nibabel, 2016, Online. doi:10.5281/zenodo.60808.

[7] N. Burgos, M. Cardoso, K. Thielemans, M. Modat, S. Pedemonte, J. Dickson, A. Barnes, R. Ahmed, J. Mahoney, J. Schott, J. Duncan, D. Atkinson, S. Arridge, B. Hutton, S. Ourselin, Attenuation correction synthesis for hybrid PET-MR scanners: Application to brain studies, Med. Imaging IEEE Trans. 33 (12) (2014) 2332–2341.

[8] M. Cardoso, M. Clarkson, M. Modat, S. Ourselin, NiftySeg: open-source software for medical image segmentation, label fusion and cortical thickness estimation, in: Proceedings of the ISBI Workshop on Open Source Medical Image Analysis Software, 2012.

[9] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, E. Shelhamer, cuDNN: efficient primitives for deep learning, arXiv:1410.0759v3.

[10] F. Chollet, et al., Keras, 2015, https://github.com/fchollet/keras.

[11] Ö. Çiçek, A. Abdulkadir, S.S. Lienkamp, T. Brox, O. Ronneberger, 3D U-net: learning dense volumetric segmentation from sparse annotation, in: Proceedings of the MICCAI, Springer, 2016, pp. 424–432.

[12] M.J. Clarkson, G. Zombori, S. Thompson, J. Totz, Y. Song, M. Espak, S. Johnsen, D. Hawkes, S. Ourselin, The NifTK software platform for image-guided inter-

ventions: platform overview and NiftyLink messaging, Int. J. Comput. Assist. Radiol. Surg. 10 (3) (2015) 301–316.
[13] R. Collobert, K. Kavukcuoglu, C. Farabet, Torch7: A MATLAB-like environment for machine learning, in: Proceedings of the NIPS Workshop on Algorithms, Systems, and Tools for Learning at Scale (Big Learning), in: EPFL-CONF-192376, 2011.
[14] A.M. Dale, B. Fischl, M.I. Sereno, Cortical surface-based analysis:I. Segmentation and surface reconstruction, Neuroimage 9 (2) (1999) 179–194.
[15] S. Dieleman, J. Schlter, C. Raffel, E. Olson, S.K. Snderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J.D. Fauw, M. Heilman, D.M. de Almeida, B. McFee, H. Weideman, G. TakÃ!'cs, P. de Rivaz, J. Crall, G. Sanders, K. Rasul, C. Liu, G. French, J. Degrave, Lasagne: first release, 2015, 10.5281/zenodo.27878
[16] T. Doel, D.I. Shakir, R. Pratt, M. Aertsen, J. Moggridge, E. Bellon, A.L. David, J. Deprest, T. Vercauteren, S. Ourselin, GIFT-Cloud: A data sharing and collaboration platform for medical imaging research, Comput. Methods Programs Biomed. 139 (2017) 181–190.
[17] H. Dong, A. Supratak, L. Mai, F. Liu, A. Oehmichen, S. Yu, Y. Guo, TensorLayer: a versatile library for efficient deep learning development, ACM Multimed, 2017.
[18] L. Fidon, W. Li, L.C. Garcia-Peraza-Herrera, J. Ekanayake, N. Kitchen, S. Ourselin, T. Vercauteren, Generalised Wasserstein Dice score for imbalanced multi-class segmentation using holistic convolutional networks, (2017). arXiv:1707.00478.
[19] B. Fischl, M.I. Sereno, A.M. Dale, Cortical surface-based analysis: II: inflation, flattening, and a surface-based coordinate system, Neuroimage 9 (2) (1999) 195–207.
[20] E. Gamma, J. Vlissides, R. Johnson, R. Helm, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.
[21] L.C. Garcia-Peraza-Herrera, W. Li, L. Fidon, C. Gruijthuijsen, A. Devreker, G. Attilakos, J. Deprest, E.V. Poorten, D. Stoyanov, T. Vercauteren, S. Ourselin, ToolNet: holistically-nested real-time segmentation of robotic surgical tools, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, 2017, pp. 5717–5722, doi:10.1109/IROS.2017.8206462.
[22] E. Gibson, F. Giganti, Y. Hu, E. Bonmati, S. Bandula, K. Gurusamy, B. Davidson, S.P. Pereira, M.J. Clarkson, D.C. Barratt, Automatic multi-organ segmentation on abdominal CT with dense v-networks, IEEE Trans. Med. Imaging (2017). in press
[23] E. Gibson, F. Giganti, Y. Hu, E. Bonmati, S. Bandula, K. Gurusamy, B.R. Davidson, S.P. Pereira, M.J. Clarkson, D.C. Barratt, Towards image-guided pancreas and biliary endoscopy: automatic multi-organ segmentation on abdominal CT with dense dilated networks, in: Proceedings of the 20th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI), 2017.
[24] E. Gibson, M.R. Robu, S. Thompson, P.E. Edwards, C. Schneider, K. Gurusamy, B. Davidson, D.J. Hawkes, D.C. Barratt, M.J. Clarkson, Deep residual networks for automatic segmentation of laparoscopic videos of the liver, in: Proceedings of the SPIE, Medical Imaging, vol. 10135, 2017, doi:10.1117/12.2255975. 101351M
[25] I. Goodfellow, NIPS 2016 tutorial: generative adversarial networks (2016). arXiv:1701.00160v4.
[26] Y. Hu, E. Gibson, L.-L. Lee, W. Xie, D.C. Barratt, T. Vercauteren, J.A. Noble, Freehand ultrasound image simulation with spatially-conditioned generative adversarial networks, in: Proceedings of MICCAI Workshop on Reconstruction and Analysis of Moving Body Organs (RAMBO), 2017.
[27] S. Ioffe, Batch renormalization: towards reducing minibatch dependence in batch-normalized models, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, 2017, pp. 1942–1950.
[28] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM International Conference on Multimedia (ACMMM), ACM, 2014, pp. 675–678.
[29] S.F. Johnsen, Z.A. Taylor, M.J. Clarkson, J. Hipwell, M. Modat, B. Eiben, L. Han, Y. Hu, T. Mertzanidou, D.J. Hawkes, S. Ourselin, NiftySim: a GPU-based nonlinear finite element package for simulation of soft tissue biomechanics, Int. J. Comput. Assist. Radiol. Surg. 10 (7) (2015) 1077–1095.
[30] K. Kamnitsas, C. Ledig, V.F. Newcombe, J.P. Simpson, A.D. Kane, D.K. Menon, D. Rueckert, B. Glocker, Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation, Med. Image Anal. 36 (2017) 61–78.
[31] S. Klein, M. Staring, K. Murphy, M.A. Viergever, J.P. Pluim, Elastix: a toolbox for intensity-based medical image registration, IEEE Trans. Med. Imaging 29 (1) (2010) 196–205.
[32] B. Landman, Z. Xu, J.E. Igelsias, M. Styner, T.R. Langerak, A. Klein, Multi-atlas labeling beyond the cranial vault, 2015, URL: https://www.synapse.org/#!Synapse:syn3193805, accessed July 2017. 10.7303/syn3193805.
[33] W. Li, G. Wang, L. Fidon, S. Ourselin, M.J. Cardoso, T. Vercauteren, On the compactness, efficiency, and representation of 3D convolutional networks: Brain parcellation as a pretext task, in: Proceedings of Information Processing in Medical Imaging (IPMI), 2017, pp. 348–360.
[34] G. Litjens, T. Kooi, B.E. Bejnordi, A.A.A. Setio, F. Ciompi, M. Ghafoorian, J.A.W.M. van der Laak, B. van Ginneken, C.I. Sánchez, A survey on deep learning in medical image analysis, Medical Image Analysis 42 (2017) 60–88, doi:10.1016/j.media.2017.07.005.
[35] S.-C. Lo, S.-L. Lou, J.-S. Lin, M.T. Freedman, M.V. Chien, S.K. Mun, Artificial convolution neural network techniques and applications for lung nodule detection, IEEE Trans. Med. Imaging 14 (4) (1995) 711–718.
[36] B.C. Lowekamp, D.T. Chen, L. Ibáñez, D. Blezek, The design of SimpleITK, Front. Neuroinf. 7 (2013).
[37] D. Mané, et al., TensorBoard: TensorFlow's visualization toolkit, 2015, https://github.com/tensorflow/tensorboard.
[38] A. Mehrtash, M. Pesteie, J. Hetherington, P.A. Behringer, T. Kapur, W.M. Wells III, R. Rohling, A. Fedorov, P. Abolmaesumi, DeepInfer: Open-source deep learning deployment toolkit for image-guided therapy, in: Proceedings of the SPIE, Medical Imaging, 10135, NIH Public Access, 2017.
[39] B.H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M.A. Weber, T. Arbel, B.B. Avants, N. Ayache, P. Buendia, D.L. Collins, N. Cordier, J.J. Corso, A. Criminisi, T. Das, H. Delingette, Ç. Demiralp, C.R. Durst, M. Dojat, S. Doyle, J. Festa, F. Forbes, E. Geremia, B. Glocker, P. Golland, X. Guo, A. Hamamci, K.M. Iftekharuddin, R. Jena, N.M. John, E. Konukoglu, D. Lashkari, J.A. Mariz, R. Meier, S. Pereira, D. Precup, S.J. Price, T.R. Raviv, S.M.S. Reza, M. Ryan, D. Sarikaya, L. Schwartz, H.C. Shin, J. Shotton, C.A. Silva, N. Sousa, N.K. Subbanna, G. Szekely, T.J. Taylor, O.M. Thomas, N.J. Tustison, G. Unal, F. Vasseur, M. Wintermark, D.H. Ye, L. Zhao, B. Zhao, D. Zikic, M. Prastawa, M. Reyes, K.V. Leemput, The multimodal brain tumor image segmentation benchmark (BraTS), IEEE Trans. Med. Imaging 34 (10) (2015) 1993–2024.
[40] F. Milletari, N. Navab, S.-A. Ahmadi, V-Net: Fully convolutional neural networks for volumetric medical image segmentation, in: Proceedings of the Fourth International Conference on 3D Vision (3DV), 2016, pp. 565–571.
[41] M. Mirza, S. Osindero, Conditional generative adversarial nets. Proc. NIPS 2016 Workshop on Adversarial Training. 2016 (2014), arXiv:1411.1784.
[42] M. Modat, G.R. Ridgway, Z.A. Taylor, M. Lehmann, J. Barnes, D.J. Hawkes, N.C. Fox, S. Ourselin, Fast free-form deformation using graphics processing units, Comput. Methods Progr. Biomed. 98 (3) (2010) 278–284.
[43] M. Nolden, S. Zelzer, A. Seitel, D. Wald, M. Müller, A.M. Franz, D. Maleike, M. Fangerau, M. Baumhauer, L. Maier-Hein, K.H. Maier-Hein, H.-P. Meinzer, I. Wolf, The medical imaging interaction toolkit: challenges and advances, Int. J. Comput. Assist. Radiol. Surg. 8 (4) (2013) 607–620.
[44] S. Pieper, B. Lorensen, W. Schroeder, R. Kikinis, The NA-MIC kit: ITK, VTK, pipelines, grids and 3D Slicer as an open platform for the medical image computing community, in: Proceedings of the IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI), 2006, pp. 698–701.
[45] T. Preston-Werner, Semantic versioning, Technical Report, T. Preston-Werner, 2015. URL: http://semver.org/.
[46] M. Reynolds, et al., Sonnet, 2017, https://github.com/deepmind/sonnet.
[47] H.R. Roth, A. Farag, E.B. Turkbey, L. Lu, J. Liu, R.M. Summers, Data from TCIA pancreas-CT, The Cancer Imaging Archive, 1994, doi:10.7937/K9/TCIA.2016.tNB1kqBU.
[48] H.R. Roth, L. Lu, A. Seff, K.M. Cherry, J. Hoffman, S. Wang, J. Liu, E. Turkbey, R.M. Summers, A new 2.5D representation for lymph node detection using random sets of deep convolutional neural network observations, in: Proceedings of the MICCAI, 2014, doi:10.1007/978-3-319-10404-1_65.
[49] T. Salimans, D.P. Kingma, Weight normalization: a simple reparameterization to accelerate training of deep neural networks (2016). arXiv:1602.07868v3.
[50] F. Seide, A. Agarwal, CNTK: Microsoft's open-source deep-learning toolkit, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, p. 2135.
[51] D. Shen, G. Wu, H.-I. Suk, Deep learning in medical image analysis, Ann. Rev. Biomed. Eng. (19) (2017) 221–248, doi:10.1146/annurev-bioeng-071516-044442.
[52] S.M. Smith, M. Jenkinson, M.W. Woolrich, C.F. Beckmann, T.E. Behrens, H. Johansen-Berg, P.R. Bannister, M. De Luca, I. Drobnjak, D.E. Flitney, R. Niazy, J. Saunders, J. Vickers, Y. Zhang, N. De Stefano, J. Brady, P. Matthews, Advances in functional and structural MR image analysis and implementation as FSL, Neuroimage 23 (2004) S208–S219.
[53] C.H. Sudre, W. Li, T. Vercauteren, S. Ourselin, M.J. Cardoso, Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations, in: Proceedings of MICCAI Workshop on Deep Learning in Medical Image Analysis (DLMIA), 2017.
[54] A. Vedaldi, K. Lenc, MatConvNet – convolutional neural networks for MATLAB, in: Proceedings of the ACMM, 2015.
[55] G. Wang, W. Li, S. Ourselin, T. Vercauteren, Automatic brain tumor segmentation using cascaded anisotropic convolutional neural networks, Proc. Multimodal Brain Tumor Segmentation (BRATS) Challenge 2017 - MICCAI workshop. 2016 (2017). arXiv:1709.00382.
[56] G. Wang, M.A. Zuluaga, W. Li, R. Pratt, P.A. Patel, M. Aertsen, T. Doel, A.L. David, J. Deprest, S. Ourselin, T. Vercauteren, DeepIGeoS: a deep interactive geodesic framework for medical image segmentation, (2017). arXiv:1707.00652v1.
[57] X. Zhou, T. Ito, R. Takayama, S. Wang, T. Hara, H. Fujita, Three-dimensional CT image segmentation by combining 2D fully convolutional network with 3D majority voting, in: Proceedings of the LABELS, Springer, 2016, pp. 111–120, doi:10.1007/978-3-319-46976-8_12.