



Published in final edited form as:

*Neuroinformatics*. 2021 January ; 19(1): 127–140. doi:10.1007/s12021-020-09477-5.

## DeepNeuro: an open-source deep learning toolbox for neuroimaging

Andrew Beers<sup>1</sup>, James Brown<sup>1</sup>, Ken Chang<sup>1</sup>, Katharina Hoebel<sup>1</sup>, Jay Patel<sup>1</sup>, K. Ina Ly<sup>1,3</sup>, Sara M. Tolaney<sup>2</sup>, Priscilla Brastianos<sup>3</sup>, Bruce Rosen<sup>1</sup>, Elizabeth R. Gerstner<sup>1,3</sup>, Jayashree Kalpathy-Cramer<sup>1</sup>

<sup>1</sup>Athinoula A. Martinos Center for Biomedical Imaging, Massachusetts General Hospital, Charlestown, MA, USA

<sup>2</sup>Department of Medical Oncology, Dana-Farber Cancer Institute, Boston MA, USA

<sup>3</sup>Division of Neuro-Oncology, Massachusetts General Hospital, Harvard Medical School, Boston MA, USA

### Abstract

Translating deep learning research from theory into clinical practice has unique challenges, specifically in the field of neuroimaging. In this paper, we present DeepNeuro, a Python-based deep learning framework that puts deep neural networks for neuroimaging into practical usage with a minimum of friction during implementation. We show how this framework can be used to design deep learning pipelines that can load and preprocess data, design and train various neural network architectures, and evaluate and visualize the results of trained networks on evaluation data. We present a way of reproducibly packaging data pre- and postprocessing functions common in the neuroimaging community, which facilitates consistent performance of networks across variable users, institutions, and scanners. We show how deep learning pipelines created with

---

Terms of use and reuse: academic research for non-commercial purposes, see here for full terms. <http://www.springer.com/gb/open-access/authors-rights/aam-terms-v1>

Correspondence: Jayashree Kalpathy-Cramer, [kalpathy@nmr.mgh.harvard.edu](mailto:kalpathy@nmr.mgh.harvard.edu).

<sup>7</sup>Author Contributions

AB, with contributions from JB and KC, was the original creator of the DeepNeuro package and all of its component parts. KH and JP contributed to modules within DeepNeuro, as well as additional data processing features. EG conducted clinical trials at Massachusetts General Hospital, contributed to data organization for clinical datasets, and was consulted on matters pertaining to clinical trials during DeepNeuro development. KIL contributed to the evaluation of results of trained network algorithms. ST and PB conducted clinical trials for brain metastases, supporting the development of DeepNeuro's segmentation modules. BR and JKC guided the conceptual development of the package, especially with regards to clinical end-users. All the authors read, reviewed, and approved the manuscript.

<sup>10</sup>Information Sharing Statement

DeepNeuro is open-source, free, and available at <https://github.com/QTIMLab/DeepNeuro> (RRID:SCR 016911). DeepNeuro makes use of the following packages via Python wrappers: 3D Slicer (RRID:SCR 005619), ANTS (RRID:SCR 004757), and dcmqi (RRID:SCR 016933). DeepNeuro depends on the following third-party Python libraries: tqdm, scikit-image, scipy (RRID:SCR 008058), numpy, pydot, matplotlib (RRID:SCR 008624), imageio, six, pyyaml, nibabel (RRID:SCR 002498), pynrrd, and pydicom (RRID:SCR 002573) [van der Walt et al., 2014, Jones et al., 2014, Hunter, 2007].

<sup>9</sup>Conflict of Interest Statement

The other authors declare no competing interests.

**Publisher's Disclaimer:** This Author Accepted Manuscript is a PDF file of a an unedited peer-reviewed manuscript that has been accepted for publication but has not been copyedited or corrected. The official version of record that is published in the journal is kept up to date and so may therefore differ from this version.

DeepNeuro can be concisely packaged into shareable Docker and Singularity containers with user-friendly command-line interfaces.

## Keywords

neuroimaging; deep learning; preprocessing; augmentation; docker

---

## 1 Introduction

Deep learning is an increasingly popular machine learning technique that involves training neural networks to learn complex, hierarchical feature representations from raw data. With the advent of more powerful graphical processing units, more efficient implementation algorithms, and more flexible coding frameworks, deep learning has become the standard approach for many computer vision tasks, and is increasingly used in speech and text analysis tasks [LeCun et al., 2015, Krizhevsky et al., 2012, Collobert and Weston, 2008, Hinton et al., 2012]. Because of its useful properties in computer vision and natural language processing tasks, deep learning is also gaining popularity in medical applications, where medical imaging and electronic health records are a key component of medical workflows. Deep learning has been successfully applied to the automated diagnosis of skin cancer, macular degeneration, diabetic retinopathy, and retinopathy of prematurity [Esteva et al., 2017, Lee et al., 2017, Gulshan et al., 2016, Brown et al., 2018]. Specifically within the field of neuroimaging, deep learning has been shown to be a particularly effective method for the automatic assessment of pathologies such as Alzheimer’s disease, stroke, glioma, and schizophrenia [Liu et al., 2014, Winzeck et al., 2018, Menze et al., 2015, Bakas et al., 2017, Gheiratmand et al., 2017, Chang et al., 2018a, Chang et al., 2019]. In each of these clinical spaces, deep learning has the potential for algorithms on certain tasks to reach or exceed accuracy and efficiency previously thought to be limited to human operators.

Scholarship in neuroimaging applications of deep learning has increased every year, and such algorithms can be expected to target an increasingly diverse array of clinical problems [Miotto et al., 2017]. Despite this progress, there remain practical challenges that prevent the widespread translation of published algorithms into research and clinical practice. Deep learning algorithms and neural networks are often fully-described in academic research, but the source code for implementations of these algorithms are seldom made available to other clinicians and researchers. When source code is made available, documentation is either absent, or presumes a level of familiarity with deep learning software beyond the expertise of the typical neuroimaging researcher. Even if source code for deep learning algorithms is both public and well-documented, operating system requirements and dependencies on other software packages may make their usage impractical without extensive technical support. Each of these failure points adds friction to the process of translating academic deep learning discoveries into research practice, and further delays the point at which deep learning scholarship can be evaluated within a clinical setting.

Provided that these software challenges are overcome, the unique nature of medical imaging data produces additional barriers to the neuroscience researcher. Medical images often

require numerous, highly-specialized preprocessing techniques to account for differences between scanners and imaging sequences, each of which can unpredictably affect the performance of deep learning algorithms. Medical imaging data is often higher resolution (as with digital pathology) or in higher dimensions (as with magnetic resonance (MR) imaging data) than traditional datasets in computer vision. As a result, images may need to be divided into patches, slices, or other representations to be computationally tractable in deep learning algorithms, and the specific implementation of these methods can have a significant impact on those algorithms' performance. Postprocessing techniques, particularly for segmentation algorithms, can have a significant effect on an neural network's accuracy [Kamnitsas et al., 2017b]. All of these problems are even more highly pronounced in the field of neuroimaging, which often features sequence-specific or diseasespecific processing steps for medical data that can only be used via separate and unrelated software packages. Even when such processing steps are documented or described in published neuroimaging research, a subtle change in their implementation by a package's author can have significant effects on the accuracy and consistency of a downstream deep learning algorithm [Kamnitsas et al., 2017a].

To address these challenges, we present DeepNeuro, a Python deep learning framework that is built to put deep learning algorithms for neuroimaging into practical usage with a minimum of friction. We will show how this framework can be used to both design and train new neural network architectures, as well as modify state-of-the-art architectures in a flexible and intuitive way. We will display how pre- and postprocessing functions common in the medical imaging community can be reproducibly employed in DeepNeuro to ensure consistent performance of networks across variable users, institutions, and scanners. We will show how pipelines created in DeepNeuro can be concisely packaged into shareable Docker and Singularity containers and command-line interfaces using DeepNeuro's pipelining resources. With these features combined, our goal is to present a package that both simplifies the production of new deep learning algorithms in neuroimaging, and enhances the reproducibility of existing algorithms.

## 2 Package Architecture

DeepNeuro is a Python package that can be installed directly via Python's default *pip* package installer, or run from within preconfigured Docker and Singularity containers. It has two primary use cases. Its first use case is as a templating language for designing pipelines that load data, apply data processing steps, train deep learning models, and evaluate these models on new data. Its second use case is as a model deployment system, in which pre-trained models and all their required pre- and postprocessing steps are packaged into simple command line options run from requirement-agnostic Docker and Singularity containers. Both of these use cases are achieved via a collection of subclasses implemented in DeepNeuro's submodules.

### 2.1 Data processing with *DataCollections*

The DeepNeuro workflow is centered on a Python class called a *DataCollection*. *DataCollections* are Python objects that store metadata about any data to be input into

subsequent processing steps, such as data augmentation, model training, or inference (Figure 1). Their purpose is to seamlessly manage the transformations required to transition data stored on disk into preprocessed and preformatted in-memory data ready to be input to a neural network. *DataCollections* have flexible data loaders to associate and, if necessary, combine different medical imaging inputs (e.g. different sequences, modalities) into NumPy arrays with simple structures [Walt et al., 2011]. They can load data from regular folder structures, regular expressions on filepaths, Hierarchical Data Format (HDF5) files, or a list of filepaths contained in text files. *DataCollections* store data as key-value pairs, associating provided filepaths or other identifiers with collections of data that may contain images, class labels, or other array-like information. *DataCollections* have the ability to save preprocessed datapoints back to disk in individual files, or aggregated into one HDF5 file. After writing, *DataCollections* can be seamlessly reloaded from HDF5 format for usage in subsequent processing pipelines.

The primary data that *DataCollections* store are Python objects called *DataGroups*. *DataGroups* represent different user-specified data inputs in a *DataCollection* that may be in different formats, such as imaging data, timeseries data, or classification labels. For a typical deep learning pipeline, one usually specifies two *DataGroups*: one for input data to the neural network, and one for ground truth that the network can be evaluated against. However, more and different groups can be specified for more complex architectures, such as for a neural network that outputs image segmentations and image classification labels simultaneously. Unique preprocessing and data augmentation pipelines can be specified for each *DataGroup*, which enables *DataCollections* to effectively process multi-formatted data streams of scanner data, classification labels, and image segmentations. The naming and structure of *DataGroups* can be recovered when *DataCollections* are loaded from HDF5 format.

## 2.2 Preprocessors and Postprocessors

**2.2.1 Preprocessing**—Standardized preprocessing methods are essential to deep learning pipelines that operate on medical images, as slight differences in preprocessing methods can lead to catastrophic prediction failures. DeepNeuro allows the user to preprocess data before model training and inference using the DeepNeuro class *Preprocessor*. The purpose of a *Preprocessor* object is to create chained pipelines of data transformations that can be selectively applied to different *DataGroups* in a *DataCollection* object (Figure 2). Transformations applied in *Preprocessor* objects can be pure Python implementations, the outputs of other pretrained neural networks in DeepNeuro, or wrappers around external programs such as 3DSlicer or ANTs [Fedorov et al., 2012, Avants et al., 2014]. These transformations can be applied to data already loaded into in memory or dynamically loaded from filepaths on disk, and can be output either as in-memory NumPy arrays or written directly back to disk. Current *Preprocessor* objects available include 3D image registration (3DSlicer), 3D image resampling (3DSlicer), N4 Bias Correction (3DSlicer, ANTs), and a brain extraction model trained using DeepNeuro (Figure 2).

**2.2.2 Postprocessing**—*Postprocessor* objects share the same structure and capabilities as *Preprocessor* objects, but are applied to *DeepNeuroModel* objects (described below)

instead of *DataCollections*. *Postprocessors* are used to apply transformations to data that has been generated by a model, rather than data that is loaded from disk. *Postprocessors* include island-removal and hole-filling for binary segmentation outputs, binarization for continuously-valued outputs, reformatting operations for one-hot encoded data, and operations to calculate common error statistics for evaluating deep learning models.

### 2.3 Data augmentation

Data augmentation is a method to increase the effective size of a dataset fed into machine learning models via spatial or intensity-based data transformations. Data augmentation is especially important in medical imaging, as medical image datasets tend to be far smaller than datasets of natural images [Hussain et al., 2017]. DeepNeuro performs augmentation via chained *Augmentation* objects, which can be applied directly to *DataCollection* objects, and are processed sequentially after all *Preprocessor* objects have been applied. Each *Augmentation* object specifies a data transformation to be applied to *DataGroups* within a *DataCollection*, and can be applied before saving data to disk (pre-augmentation), or during training (lazy augmentation). Augmentations can also be applied to data before evaluation on trained models, to see how data transformations affect the accuracy of a given model, or for averaging a model's predictions.

Current *Augmentation* objects available include 2D and 3D flips and rotations, intensity scaling and shifting, 2D and 3D patch extraction, random data masking, and nearest-neighbor downsampling. Patch extraction can be performed to preferentially select patches that match certain criteria, such as being near a tissue or structure of interest (Figure 4).

### 2.4 Model design with *DeepNeuroModels*

A Python programmer wishing to create a neural network has many available tools. For example, Keras, one of the most popular deep learning frameworks, is an abstracted language for constructing neural networks that can be run with several different backends, including TensorFlow, Theano, and MXNet [Abadi et al., 2016, Bergstra et al., 2010, Chen et al., 2015]. Other deep learning toolboxes exist outside of the Keras framework, such as the popular deep learning framework PyTorch [Paszke et al., 2017] and more recently the programming language Julia [Bezanson et al., 2017]. Inadvertently, this wide variety of languages has created difficulty in making models open-source in practice, as researchers split between TensorFlow, PyTorch, Julia, or another framework may have difficulty readily implementing each others' code.

The *DeepNeuroModel* is an abstracted model class designed to address this problem within the practical context of neuroimaging. *DeepNeuroModels* are objects that take *DataCollections* as inputs, and process them in accordance with a minimal set of standardized functions that are shared across all popular deep learning frameworks. These include model training, model saving, loading pre-trained models, generating model callbacks for training loss and other statistics, and performing inference and model evaluation. This minimum set of functions is currently implemented via subclasses for Keras and TensorFlow, and can be called agnostic of either framework from DeepNeuro's

interface. These functions take advantage of TensorFlow's existing model-training algorithms for efficient implementation on CPU and GPU computing systems alike.

Users can customize *DeepNeuroModel* objects by feeding in different architecture parameters. Some of these parameters affect the overall structure of the network, such as the number of pooling layers, the overall depth of the network, or the model dimensionality (2D or 3D). Other parameters affect fixed design choices in a given model, such as filter numbers, dropout ratios, the presence of batch normalization (to reduce overfitting), the choice of activation functions, the choice of cost functions, kernel size, and stride size. Parameters specific to training regimes can also be specified, including batch sizes, learning rates and learning rate schedules, and optimizers. Training callbacks can be specified for events like model error logging, model inference on validation or testing datasets, and early stopping after model performance has plateaued.

**2.4.1 Model implementations**—For segmentation applications, we have implemented the U-Net architecture in both 2D and 3D [Ronneberger et al., 2015, Çiçek et al., 2016]. For image synthesis applications, we have implemented both a traditional generative adversarial network architecture (GAN) and the progressively growing GAN architecture proposed by Karras et al. [Goodfellow et al., 2014, Karras et al., 2017]. For image classification applications, we have included standard implementations of InceptionNet, VGGNet, and the ResNet, with architectures and pretrained weights imported from Keras. [Szegedy et al., 2015, Simonyan and Zisserman, 2014, He et al., 2016] *DeepNeuroModels* can currently be easily extended to models already developed by users in TensorFlow and Keras using the *TensorflowModel* and *KerasModel* subclasses respectively.

**2.4.2 Model inference and custom outputs**—Users can run inference on a *DeepNeuroModel* and create other custom outputs via *Output* objects. The most relevant *Output* class for a deep learning model is *Inference*, which computes the output of a *Model* after being fed input data. Different subclasses of *Inference* are available for different deep learning tasks. *ClassInference* objects, for example, provide class pseudo-probabilities for each output class of a trained model. *PatchesInference* objects allow users to run segmentation models that predict on a patch-wise basis by taking in input images, splitting them into component patches, predicting on each patch, and stitching together the predicted patches for an output segmentation (Figure 3). *GANInference* and *GANInterpolation* objects allow users to generate synthetic images from trained models, and to explore the latent space from which these images are drawn.

## 2.5 Additional Utilities

**2.5.1 Converters for medical image data formats**—Data loading features for NIfTI (Neuroimaging Informatics Technology Initiative), Nrrd (Nearly raw raster data), and DICOM (Digital Imaging and Communications in Medicine) file formats are also included as standalone functions, as well as data saving functions for both NIfTI and DICOM Segmentation Objects (DSO) via wrappers on the external package *dcmqi* [Larobina and Murino, 2014, Kindlmann et al., 2008, Clunie, 2000, Herz et al., 2017].

**2.5.2 Data visualization**—DeepNeuro includes a multi-purpose *visualize* function for visualizing multidimensional data stored in *DataCollections*. Segmentation and classification data can be viewed alongside raw imaging data, and data can be viewed after having been transformed by *Preprocessor* and *Augmentation* objects (Figure 4). Visualizations can be produced periodically during model training to visualize training progress. These visualizations can be aggregated into animations or tiled mosaic images for qualitative analysis of model overfitting.

### 3 Pipeline distribution

Upon finishing a model training pipeline with DeepNeuro, users may wish to share their pipelines with others in a reproducible fashion. DeepNeuro facilitates model-sharing via a set of templating functions in the *pipelines* module that help users turn their trained models into easy-to-use command-line tools, and further helps them distribute their work via Docker and Singularity containers. Docker containers are objects akin to virtual machines that efficiently install all dependencies and data required to run a certain piece of software, and can be run with trivial start-up time on any operating system [Merkel, 2014]. Singularity is a similar containerization system that is cross-compatible with Docker, but offers additional security features that may be desirable in large computing clusters [Kurtzer et al., 2017]. DeepNeuro Docker containers, and by extension Singularity containers, are based on the *nvidia-docker*, which allows users to take advantage of NVIDIA GPUs for faster model inference. They also come pre-loaded with all of the Python packages and optional external software packages that one may require for specialized functions in DeepNeuro.

The *pipelines* module contains several utilities for easily turning your DeepNeuro pipelines into command-line interfaces with a minimum of prior coding experience, and utility functions for easily creating the configuration files necessary to create pipeline-specific Docker and Singularity containers. The *pipelines* module of DeepNeuro also contains several example pipelines (detailed in “Sample Applications”) that exhibit a framework for building DeepNeuro training and inference modules and Docker containers, and templating functions for quickly building out the files necessary for one’s own pipeline. Models approved for public distribution by DeepNeuro’s maintainers are stored outside of DeepNeuro in a cloud-based data management system, and can be downloaded via the *load* module in DeepNeuro, or acquired pre-downloaded via Docker and Singularity containers.

#### 3.1 Publicly-available modules

DeepNeuro modules are currently available for brain extraction in conventional MR, enhancing tumor and whole tumor segmentation in glioblastoma conventional MR, enhancing tumor segmentations in brain metastases conventional MR images, and ischemic stroke segmentation on diffusion-weighted MR imaging (Figure 5). Docker and Singularity containers exist for each module, containing the base DeepNeuro container and the models necessary to run that particular module. Instructions for how to run each model is publicly available on DeepNeuro’s Github page.

## 4 Example Usage: Brain Metastases Segmentation

To demonstrate the practical usage of DeepNeuro, we show how to construct a typical pipeline in the use case of brain metastases segmentation. A clinician interested in such a pipeline may be interested in reducing the time they spend manually identifying such metastases, applying metastases segmentation to large retrospective datasets, or in supplementing their own annotation process by providing an algorithmically-generated first guess. To initiate creating such an algorithm, the clinician would collect standardized patient data with brain metastases during a clinical trial. They would then split these patients into those used for training a deep learning algorithm, and those used for evaluating the effectiveness of that algorithm. Data from all patients would be preprocessed simultaneously to ensure standardization before input to the algorithm.

In this use case, problems typical to the application of deep learning algorithms to neuroimaging data are encountered, such as class imbalance and image normalization, and are circumvented with *Preprocessor*, *Augmentation*, and training parameters in *DeepNeuroModels*. An end-to-end pipeline for preprocessing, model training, and evaluation on a separate test set is created using 26 lines of code, available for inspection in the Supplemental Data (Supplemental Figure 1).

### 4.1 Problem description and data loading

Our training dataset consists of 63 visits from 36 patients in an IRB-approved clinical trial for brain metastases with a diagnosis of breast cancer ([NCT02260531](#)), and 64 visits from 24 patients in an IRB-approved clinical trial for brain metastases of any histology, for a total of 60 patients and 127 visits ([NCT02886585](#)). Our testing dataset consisted of 18 visits from 8 patients in the latter clinical trial for brain metastases of any histology. For each patient visit, an expert neurooncologist with 10+ years of experience has created binary, voxelwise labelmaps to indicate the presence or absence of metastatic lesions. We store filepaths leading to all MR images and labelmaps for this dataset in commaseparated value format, and create links to these data files in DeepNeuro with *DataCollection* objects.

### 4.2 Normalizing data with brain extraction

While patients with brain metastases can be expected to receive broadly similar MR sequences during treatment, differences in scanner choice, design choices in sequence protocols, and random effects during image acquisition can all affect the precise distribution of MR values. This may manifest in differences in the distribution of MR intensities in these studies[Stonnington et al., 2008]. One approach when processing MR data is to normalize the intensities of this data to a common distribution, but this process requires accurate brain-extraction so as not to be influenced by highly heterogeneous non-brain structures that can vary between scans.

DeepNeuro can normalize the input data by applying a *SkullStripping* preprocessor object followed by a *ZeroMeanNormalization* object to the instantiated *DataCollections*. Brain extraction will be performed on each patient using a deep learning model previously trained

in DeepNeuro, and intensities will subsequently be normalized such that voxelwise intensities in the brain have a mean of 0 and a standard deviation of 1 for each input scan.

### 4.3 Augmenting data with selective image patching

Deep learning with neuroimaging can suffer from multiple sampling issues. Neuroimaging patient datasets are small relative to conventional machine learning datasets, and if processed directly, are vulnerable to overfitting. Because of this, neuroimaging data is often subdivided into 2D or 3D patches of a set size to synthetically augment the dataset size. The process of patching has the added benefit of reducing the computational burden of deep learning algorithms, leading to its widespread adoption as a default implementation for volumetric data processing.

Once data is subsampled into patches, a second sampling problem can arise in certain pathologies. Brain metastases, for example, can be found in any area of the brain, but individually occupy a small portion compared to the brain's total area. If patches are extracted from the brain randomly, metastatic tissue is likely to be drastically undersampled relative to normal brain tissue, especially in patients with fewer lesions. This can lead to difficulties during training, as models will be learning from input data without any tissues of interest [Buda et al., 2018].

One can overcome both such sampling problems with DeepNeuro's *PatchExtraction* object. By appending this object to a *DataCollection*, one can automatically subdivide their data into randomly extracted patches. In the current case, the *PatchExtraction* object will be parameterized to extract 3D cubic patches with a length of 32 voxels on each side. Furthermore, the latter problem of unbalanced class distributions can be addressed via DeepNeuro's "patch regions" functionality. The user can define two conditions that define which regions can be sampled from, and the relative ratios that regions will be sampled in, and feed this information as additional parameters into the *PatchExtraction* object. In this case, two patches regions will be defined by where the ground truth labelmap is 1 (metastatic tumor region) and where the ground truth labelmap is 0 (non-pathological region), and the former will be oversampled by a ratio of 70% to 30% for a total of 200 patches per patient. This will ensure that metastatic tumors are well-represented during training.

### 4.4 Model creation using a weighted loss function

While the previous step ensures that relatively more patches will be selected that contain a metastatic brain tumor, some lesions are so small that there may be additional class imbalance problems on the level of the individual image patch. For example, a given metastatic lesion may have as few as 100 voxels labeled out of a possible 32,768 voxels in a single patch, leading to misleadingly high raw accuracy measures (<99%) in a patch predicted to have no metastatic tissue.

To address such class imbalance problems, DeepNeuro provides users with a variety of cost functions better suited to datasets with imbalanced classes. One particularly pertinent example is the weighted categorical cross-entropy function, which can be parameterized to penalize misclassifications of tumor tissues more highly than misclassifications of non-

tumor tissue. In this particular case, one can initially weight non-tumor misclassifications at 0.1 and tumor misclassifications at 3.0, to ensure that missed metastatic tumor voxels are penalized more heavily during training.

To create a model with such a cost function, one instantiates a *UNet* *DeepNeuro* model object. The U-Net is a popular neural network architecture that has been implemented for both 2D and 3D segmentation tasks, and has found popular usage in segmentation for medical imaging [Ronneberger et al., 2015, Çiçek et al., 2016]. To add the weighted cross entropy loss function to our model with the chosen parameters, one can set its *cost function* parameter to “weighted categorical cross entropy” and the *categorical weighting* parameter to a Python dictionary object, {0 : 0.1, 1 : 3.0}.

#### 4.5 Training with progress visualizations

Models are trained on patches, but human observers evaluate models based on their performance on an entire MR image, rather than on an individual, limited field-of-view patch. Furthermore, without effective visualization, it can be difficult to interpret the meaning of a cost function’s value as it relates to subjective quality of a tissue segmentation.

*DeepNeuro* addresses both of these needs via custom visualization callbacks during training. In this case, a *PatchesInference* object can be instantiated to run model inference on images composed from a collection of patches, rather than on randomized sets of patches individually. This object can then be appended as a custom callback to the *DeepNeuroModel* in this pipeline, and can be parameterized to save periodic whole-brain images of these predictions as the model trains. This allows users to visually inspect the accuracy of the network as it trains, and better understand tissue areas that the network struggles to segment correctly (see Figure 6).

#### 4.6 Evaluating data on multiple metrics

Different medical imaging problems may have different preferred ways to evaluate model performance. For example, automatic segmentation algorithms are commonly evaluated by the Dice coefficient, which measures how well a generated segmentation overlaps with a ground-truth expert segmentation and consequently how similar downstream analyses based on such segmentations may be. In this case, the problem of brain metastases segmentation may require additional and unique segmentation evaluations. Because individual metastases have a relatively high ratio of boundary to total volume, dice coefficient measures are likely to misleadingly low. More importantly to the clinician, the exact contours of single brain metastases may be less essential to disease prognosis than an accurate measure of the absolute number of brain metastases [Sperduto et al., 2008].

To address such concerns in the context of *DeepNeuro*, one can append a *PatchesInference* object to their *DeepNeuroModel*, and use *Postprocessor* objects to evaluate a set of pre-configured error functions on label data. To create postprocessed segmentation maps, one can append a *LabelBinarization* object, which converts the real-valued segmentation outputs of a neural network into binary segmentation maps. One can then append an *ErrorCalculation* object to calculate prediction error against a provided set of ground truth segmentations. The *ErrorCalculation* object can be parameterized to calculate the traditional

Dice coefficient score, and also a measure titled "cluster accuracy", which calculates how many unique ground-truth segmentations clusters were recovered in the generated segmentation. Calculating both of these measures allows users to understand both the voxelwise accuracy of the labelmaps, and the prognostic usage of the algorithm in determining the total amount of true metastases recovered.

## 5 Discussion

We present a Python package and model distribution system titled DeepNeuro. It is a toolset for building and training neural network architectures, a data pre- and postprocessing tool for neuroimaging, and a templating and distribution system for deep learning algorithms in neuroimaging. Using Docker and Singularity as distribution platforms, it provides access to popular and validated software packages already used by neuroimaging researchers, such as 3DSlicer, ANTs, and FSL. It integrates these tools seamlessly into a pipeline driven by popular deep learning software packages such as Keras and TensorFlow, and can perform computations with and without access to graphical processing units. It ensures the accurate reproduction of results by packaging together not only neural network architectures, but also the exact pre- and postprocessing methods required for these neural network architectures under their original training regimes.

Other open-source software packages have been developed to design and distribute deep learning models specifically for medical imaging data. NiftyNet is a software package under active development that serves as a framework and templating tool for deep learning in medical imaging datasets, as well as a model repository for individual use cases [Gibson et al., 2018]. DLTK also serves as a framework for deep learning with medical imaging, and also has a repository for trained neural networks [Pawlowski et al., 2017]. ModelHub.ai is an open-source, contributor-driven framework for sharing deep learning models and processing code created in any framework via Docker containers, and has an online interface for model testing [Imaging et al., 2018]. DeepInfer is a module that allows deep learning algorithms to be used within the context of 3DSlicer, a popular graphical platform for medical imaging used by researchers and clinicians alike [Mehrtash et al., 2017]. While all of these packages taken together provide a strong foundation for both sharing and designing deep learning algorithms, only NiftyNet and DLTK do both simultaneously, and none focus specifically on applications to neuroimaging data. In addition to these qualities, DeepNeuro also has the unique ability to wrap around medical imaging functions in external software packages, allowing users to use popular methods in the medical imaging literature without have to laboriously integrate such code into DeepNeuro themselves.

The most prominent limitation of DeepNeuro is one that is common to all deep learning frameworks: its continued development is likely to lag behind the rapid pace with which the field is advancing. For example, the developers of Keras and TensorFlow, the frameworks upon which DeepNeuro is currently based, may add additional features, or remove deprecated features, faster than the current set of DeepNeuro contributors can integrate. Similar concerns exist for neural network architectures. We attempt to implement the most successful and widely-used architectures in medical imaging, but these architectures are every day being improved with new design innovations or being supplanted by yet more

powerful forms. To the point of updating software packages, we address this first limitation through versioning via Docker and Singularity containers. While the packages that undergird DeepNeuro may update or become unsupported in the coming years, DeepNeuro will provide versions of these packages “frozen” in containers at no additional configuration time to the user. These containers can thus be run regardless of the present-day status of the packages upon which they depend. To the point of updating architectures, we intend to make regular reproductions of the network architectures found in popular academic works, and will encourage the wider DeepNeuro community to submit pull requests to DeepNeuro if they wish to integrate new modules for neural network design.

We will continue to add features to DeepNeuro, and encourage contributions from the users of DeepNeuro in the form of both features and additional modules. Moving forward, there are four areas we will forefront in DeepNeuro’s development:

- We will build a graphical user interface to construct and run deep learning models. DeepNeuro’s architecture is already heavily modularized, and lends itself well to flowchart-like representations that users could stitch together via drag-and-drop mouse operations. Our aim is to lower the barrier to entry for DeepNeuro, so that researchers and clinicians without programming experience can use this tool.
- We will expand the subclasses of *DeepNeuroModel* to include models created in PyTorch and other programming languages, like Julia, to facilitate the rapid development for users not accustomed to Tensorflow and Keras. This will also aid in the adoption of open-source models written originally in these languages.
- We plan to add support for training DeepNeuro models in a distributed fashion, such that multiple users with independent datasets can share the same model under controlled conditions [Chang et al., 2018b]. Such concerns are especially important with medical imaging data, where patient privacy restrictions often prohibit the sharing of medical data between different medical institutions.
- We plan to add additional support for the integration of DeepNeuro into high workload clinical imaging systems. This includes the ability to run DeepNeuro containers as standalone inference servers for medical imaging data, support for parallelized inference on single GPUs, parallelized model training on multiple GPUs, and the exporting of DeepNeuro inference outputs and model frameworks into web-friendly formats such as JavaScript Object Notation (JSON).

The Glioblastoma Segmentation and SkullStripping modules in DeepNeuro are already being used in a research context for the routine analysis of clinical trials at the Massachusetts General Hospital, and DeepNeuro is currently supporting active research in the fields of adrenoleukodystrophy, stroke, brain metastases, and aneurysms. We are working to integrate DeepNeuro with the clinical environment at the Massachusetts General Hospital, to facilitate training and testing deep learning models on a variety of medical use cases derived directly from hospital imaging systems. Further research priorities with DeepNeuro include studies that evaluate the relative effectiveness of DeepNeuro segmentation methods for diagnosing and forecasting disease when compared with gold standard methods.

We anticipate that DeepNeuro will address many practical problems in the new field of deep learning in neuroimaging. It greatly reduces the learning curve from general-purpose deep learning packages like TensorFlow and PyTorch, reduces the time spent on devising and writing data-loading and preprocessing code, and mitigates the need for complex informatics infrastructures and installation requirements when implementing deep learning tools. It can be used in concert with other machine learning packages for method ensembling, and integrated into existing diagnostic systems with relative ease.

## 6 Installation Instructions

DeepNeuro can be installed via several methods. One method is to install using Python's *pip* package manager using the following command:

```
pip install deepneuro
```

If installing locally via *pip*, users will not have access to external packages in DeepNeuro. To remedy this issue without installing each package individually, users can pull the DeepNeuro Docker container with DeepNeuro and all external packages pre-installed using the following command:

```
docker pull qtmlab/deepneuro
```

Singularity containers can be pulled with the following command:

```
singularity pull docker://qtmlab/deepneuro
```

## Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

## Acknowledgements

The Center for Clinical Data Science at Massachusetts General Hospital and the Brigham and Woman's Hospital provided technical and hardware support for the development of DeepNeuro, including access to high-powered graphical processing units.

### 8 Funding

This project was supported by a training grant from the NIH Blueprint for Neuroscience Research (T90DA022759/R90DA023427) and the National Institute of Biomedical Imaging and Bioengineering (NIBIB) of the National Institutes of Health under award number 5T32EB1680 to KC. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. This study was supported by National Institutes of Health grants U01 CA154601, U24 CA180927, and U24 CA180918 to JKC. This research was carried out in whole or in part at the Athinoula A. Martinos Center for Biomedical Imaging at the Massachusetts General Hospital, using resources provided by the Center for Functional Neuroimaging Technologies, P41EB015896, a P41 Biotechnology Resource Grant supported by the National Institute of Biomedical Imaging and Bioengineering (NIBIB), National Institutes of Health.

JKC is a consultant/advisory board member for Infotech, Soft. ERG is an advisory board member for Blue Earth Diagnostics. BR is on the advisory board for ARIA, Butterfly, Inc., DGMIF (Daegu-Gyeongbuk Medical Innovation Foundation), QMENTA, Subtle Medical, Inc., is a consultant for Broadview Ventures, Janssen Scientific, ECRI Institute, GlaxoSmithKline, Hyperfine Research, Inc., Peking University, Wolf Greenfield, Superconducting Systems, Inc., Robins Kaplin, LLC, Millennium Pharmaceuticals, GE Healthcare, Siemens, Quinn Emanuel Trial Lawyers, Samsung, Shenzhen Maternity and Child Healthcare Hospital, and is a founder of BLINKAI Technologies, Inc. PB is a consultant for Angiochem, Lilly, Tesaro, Genentech-Roche; has received honoraria from Genentech-Roche and Merck and has received institutional funding from Merck and Pfizer. SMT receives institutional research funding from Novartis, Genentech, Eli Lilly, Pfizer, Merck, Exelixis, Eisai, Bristol Meyers Squibb, AstraZeneca, Cyclacel, Immunomedics, Odenate, and Nektar. SMT has served as an advisor/

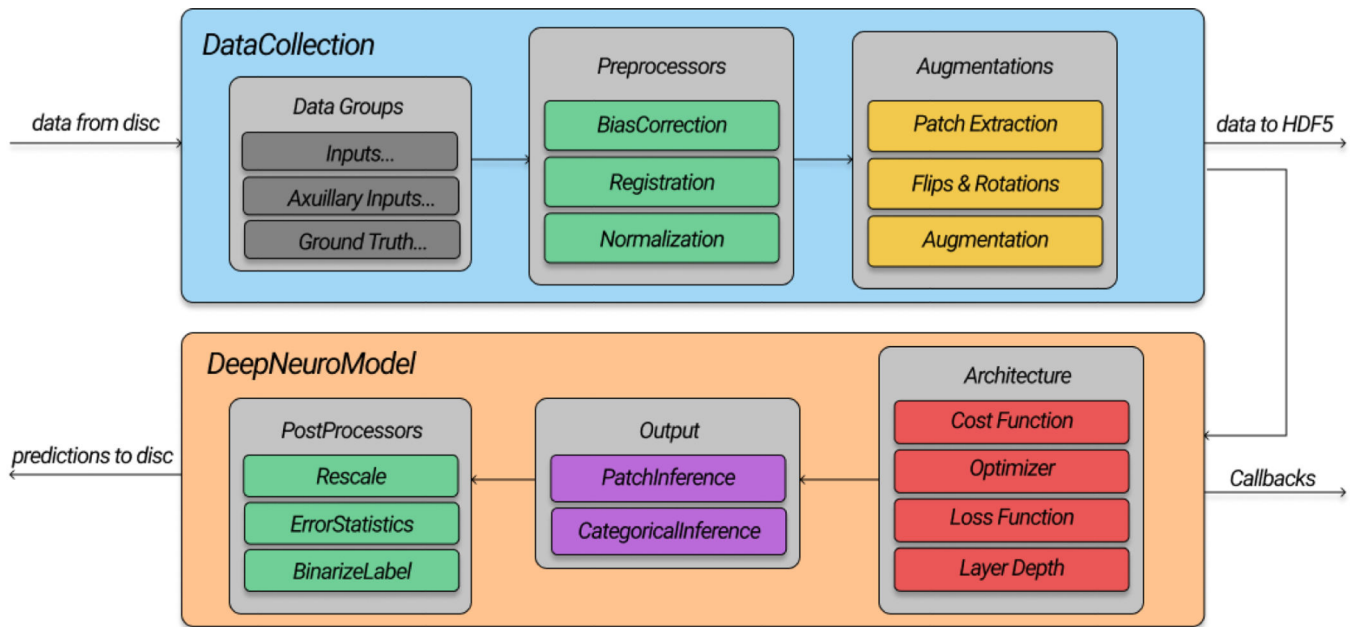
consultant to Novartis, Eli Lilly, Pfizer, Merck, AstraZeneca, Eisai, Puma, Genentech, Immunomedics, Nektar, Tesaro, and Nanostring.

## References

- Abadi et al, 2016. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al. (2016). Tensorflow: a system for large-scale machine learning. In OSDI, volume 16, pages 265–283.
- Avants et al, 2014. Avants BB, Tustison NJ, Stauffer M, Song G, Wu B, and Gee JC (2014). The insight toolkit image registration framework. *Frontiers in neuroinformatics*, 8:44. [PubMed: 24817849]
- Bakas et al, 2017. Bakas S, Akbari H, Sotiras A, Bilello M, Rozycki M, Kirby JS, Freymann JB, Farahani K, and Davatzikos C. (2017). Advancing the cancer genome atlas glioma mri collections with expert segmentation labels and radiomic features. *Scientific data*, 4:170117. [PubMed: 28872634]
- Bergstra et al, 2010. Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, Turian J, Warde-Farley D, and Bengio Y. (2010). Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, volume 1.
- Bezanson et al, 2017. Bezanson J, Edelman A, Karpinski S, and Shah VB (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.
- Brown et al, 2018. Brown JM, Campbell JP, Beers A, Chang K, Ostmo S, Chan RP, Dy J, Erdogmus D, Ioannidis S, Kalpathy-Cramer J, et al. (2018). Automated diagnosis of plus disease in retinopathy of prematurity using deep convolutional neural networks. *JAMA ophthalmology*.
- Buda et al, 2018. Buda M, Maki A, and Mazurowski MA (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Netw*, 106:249–259. [PubMed: 30092410]
- Chang et al, 2018a. Chang K, Bai HX, Zhou H, Su C, Bi WL, Agbodza E, Kavouridis VK, Senders JT, Boaro A, Beers A, et al. (2018a). Residual convolutional neural network for the determination of idh status in low-and high-grade gliomas from mr imaging. *Clinical Cancer Research*, 24(5):1073–1081. [PubMed: 29167275]
- Chang et al, 2018b. Chang K, Balachandar N, Lam C, Yi D, Brown J, Beers A, Rosen B, Rubin DL, and Kalpathy-Cramer J. (2018b). Distributed deep learning networks among institutions for medical imaging. *Journal of the American Medical Informatics Association*.
- Chang et al, 2019. Chang K, Beers AL, Bai HX, Brown JM, Ly KI, Li X, Senders JT, Kavouridis VK, Boaro A, Su C, et al. (2019). Automatic assessment of glioma burden: A deep learning algorithm for fully automated volumetric and bi-dimensional measurement. *Neuro-oncology*.
- Chen et al, 2015. Chen T, Li M, Li Y, Lin M, Wang N, Wang M, Xiao T, Xu B, Zhang C, and Zhang Z. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*.
- Çiçek et al, 2016. Çiçek Ö, Abdulkadir A, Lienkamp SS, Brox T, and Ronneberger O. (2016). 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 424–432. Springer.
- Clunie, 2000. Clunie DA (2000). DICOM structured reporting. PixelMed publishing.
- Collobert and Weston, 2008. Collobert R and Weston J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.
- Esteva et al, 2017. Esteva A, Kuprel B, Novoa RA, Ko J, Swetter SM, Blau HM, and Thrun S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115. [PubMed: 28117445]
- Fedorov et al, 2012. Fedorov A, Beichel R, Kalpathy-Cramer J, Finet J, Fillion-Robin J-C, Pujol S, Bauer C, Jennings D, Fennessy F, Sonka M, et al. (2012). 3d slicer as an image computing platform for the quantitative imaging network. *Magnetic resonance imaging*, 30(9):1323–1341. [PubMed: 22770690]

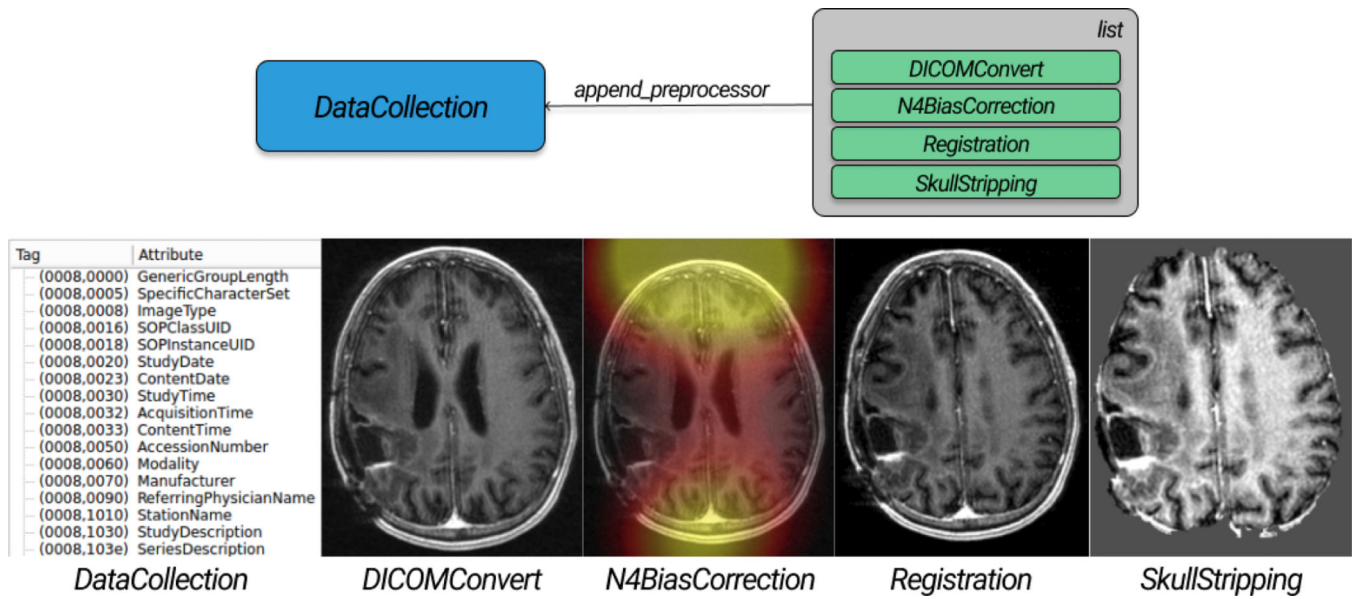
- Gheiratmand et al, 2017. Gheiratmand M, Rish I, Cecchi GA, Brown MR, Greiner R, Polosecki PI, Bashivan P, Greenshaw AJ, Ramasubbu R, and Dursun SM (2017). Learning stable and predictive network-based patterns of schizophrenia and its clinical symptoms. *NPJ schizophrenia*, 3(1):22. [PubMed: 28560268]
- Gibson et al, 2018. Gibson E, Li W, Sudre C, Fidon L, Shakir DI, Wang G, Eaton-Rosen Z, Gray R, Doel T, Hu Y, et al. (2018). Niftynet: a deep-learning platform for medical imaging. *Computer methods and programs in biomedicine*, 158:113–122. [PubMed: 29544777]
- Goodfellow et al, 2014. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, and Bengio Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.
- Gulshan et al, 2016. Gulshan V, Peng L, Coram M, Stumpe MC, Wu D, Narayanaswamy A, Venugopalan S, Widner K, Madams T, Cuadros J, et al. (2016). Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410. [PubMed: 27898976]
- He et al, 2016. He K, Zhang X, Ren S, and Sun J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- Herz et al, 2017. Herz C, Fillion-Robin J-C, Onken M, Riesmeier J, Lasso A, Pinter C, Fichtinger G, Pieper S, Clunie D, Kikinis R, et al. (2017). Dcmqi: an open source library for standardized communication of quantitative image analysis results using dicom. *Cancer research*, 77(21):e87–e90. [PubMed: 29092948]
- Hinton et al, 2012. Hinton G, Deng L, Yu D, Dahl GE, Mohamed A. r., Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97.
- Hunter, 2007. Hunter JD (2007). Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95.
- Hussain et al, 2017. Hussain Z, Gimenez F, Yi D, and Rubin D. (2017). Differential data augmentation techniques for medical imaging classification tasks. In *AMIA Annual Symposium Proceedings*, volume 2017, page 979 American Medical Informatics Association.
- Imaging et al, 2018. Imaging, C., Bioinformatics Lab at the Harvard Medical School, B. . W. H., and Institute, D.-F. C. (2018). Modelhub.ai
- Jones et al, 2014. Jones E, Oliphant T, and Peterson P. (2014). {SciPy}: open source scientific tools for {Python}.
- Kamnitsas et al, 2017a. Kamnitsas K, Baumgartner C, Ledig C, Newcombe V, Simpson J, Kane A, Menon D, Nori A, Criminisi A, Rueckert D, et al. (2017a). Unsupervised domain adaptation in brain lesion segmentation with adversarial networks. In *International Conference on Information Processing in Medical Imaging*, pages 597–609. Springer.
- Kamnitsas et al, 2017b. Kamnitsas K, Ledig C, Newcombe VF, Simpson JP, Kane AD, Menon DK, Rueckert D, and Glocker B. (2017b). Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78. [PubMed: 27865153]
- Karras et al, 2017. Karras T, Aila T, Laine S, and Lehtinen J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.
- Kindlmann et al, 2008. Kindlmann G, Bigler J, and Van Uiter D. (2008). Nrrd file format.
- Krizhevsky et al, 2012. Krizhevsky A, Sutskever I, and Hinton GE (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.
- Kurtzer et al, 2017. Kurtzer GM, Sochat V, and Bauer MW (2017). Singularity: Scientific containers for mobility of compute. *PloS one*, 12(5):e0177459. [PubMed: 28494014]
- Larobina and Murino, 2014. Larobina M and Murino L. (2014). Medical image file formats. *Journal of digital imaging*, 27(2):200–206. [PubMed: 24338090]
- LeCun et al, 2015. LeCun Y, Bengio Y, and Hinton G. (2015). Deep learning. *nature*, 521(7553):436. [PubMed: 26017442]

- Lee et al, 2017. Lee CS, Baughman DM, and Lee AY (2017). Deep learning is effective for classifying normal versus age-related macular degeneration oct images. *Ophthalmology Retina*, 1(4):322–327. [PubMed: 30693348]
- Liu et al, 2014. Liu S, Liu S, Cai W, Pujol S, Kikinis R, and Feng D. (2014). Early diagnosis of alzheimer’s disease with deep learning. In *Biomedical Imaging (ISBI), 2014 IEEE 11th International Symposium on*, pages 1015–1018. IEEE.
- Mehrtash et al, 2017. Mehrtash A, Pesteie M, Hetherington J, Behringer PA, Kapur T, Wells WM, Rohling R, Fedorov A, and Abolmaesumi P. (2017). Deepinfer: open-source deep learning deployment toolkit for image-guided therapy In *Medical Imaging 2017: Image-Guided Procedures, Robotic Interventions, and Modeling*, volume 10135, page 101351K International Society for Optics and Photonics.
- Menze et al, 2015. Menze BH, Jakab A, Bauer S, Kalpathy-Cramer J, Farahani K, Kirby J, Burren Y, Porz N, Slotboom J, Wiest R, et al. (2015). The multimodal brain tumor image segmentation benchmark (brats). *IEEE transactions on medical imaging*, 34(10):1993. [PubMed: 25494501]
- Merkel, 2014. Merkel D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.
- Miotto et al, 2017. Miotto R, Wang F, Wang S, Jiang X, and Dudley JT (2017). Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*.
- Paszke et al, 2017. Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, and Lerer A. (2017). Automatic differentiation in pytorch.
- Pawlowski et al, 2017. Pawlowski N, Ktena SI, Lee MC, Kainz B, Rueckert D, Glocker B, and Rajchl M. (2017). Dltk: State of the art reference implementations for deep learning on medical images. *arXiv preprint arXiv:1711.06853*.
- Ronneberger et al, 2015. Ronneberger O, Fischer P, and Brox T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.
- Simonyan and Zisserman, 2014. Simonyan K and Zisserman A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Sperduto et al, 2008. Sperduto PW, Berkey B, Gaspar LE, Mehta M, and Curran W. (2008). A new prognostic index and comparison to three other indices for patients with brain metastases: an analysis of 1,960 patients in the rtog database. *International Journal of Radiation Oncology\* Biology\* Physics*, 70(2):510–514.
- Stonnington et al, 2008. Stonnington CM, Tan G, Klöppel S, Chu C, Draganski B, Jack CR Jr, Chen K, Ashburner J, and Frackowiak RS (2008). Interpreting scan data acquired from multiple scanners: a study with alzheimer’s disease. *Neuroimage*, 39(3):1180–1185. [PubMed: 18032068]
- Szegedy et al, 2015. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, and Rabinovich A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.
- van der Walt et al, 2014. van der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yager N, Gouillart E, Yu T, and the scikit-image contributors (2014). *scikit-image: image processing in Python*. *PeerJ*, 2:e453. [PubMed: 25024921]
- Walt et al, 2011. Walt S. v. d., Colbert SC, and Varoquaux G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.
- Winzeck et al, 2018. Winzeck S, Hakim A, McKinley R, Pinto JAADS, Alves V, Silva C, Pisov M, Krivov E, Belyaev M, Monteiro M, et al. (2018). Isles 2016 & 2017-benchmarking ischemic stroke lesion outcome prediction based on multispectral mri. *Frontiers in Neurology*, 9:679. [PubMed: 30271370]



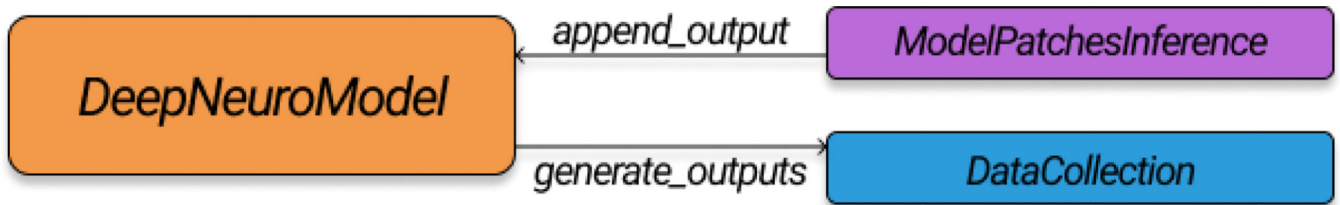
**Fig. 1.**

A diagrammatic view of DeepNeuro's architecture. Data processing operations, such as scan normalization or data augmentation, are methods applied in the context of the *DataCollection* class. A *DataCollection* class can write data out, or interact with a *DeepNeuroModel* class. *DeepNeuroModels* contain deep learning architectures, training regimes, and output methods such as patch reconstruction. Graphic created using the vector graphics program Figma.

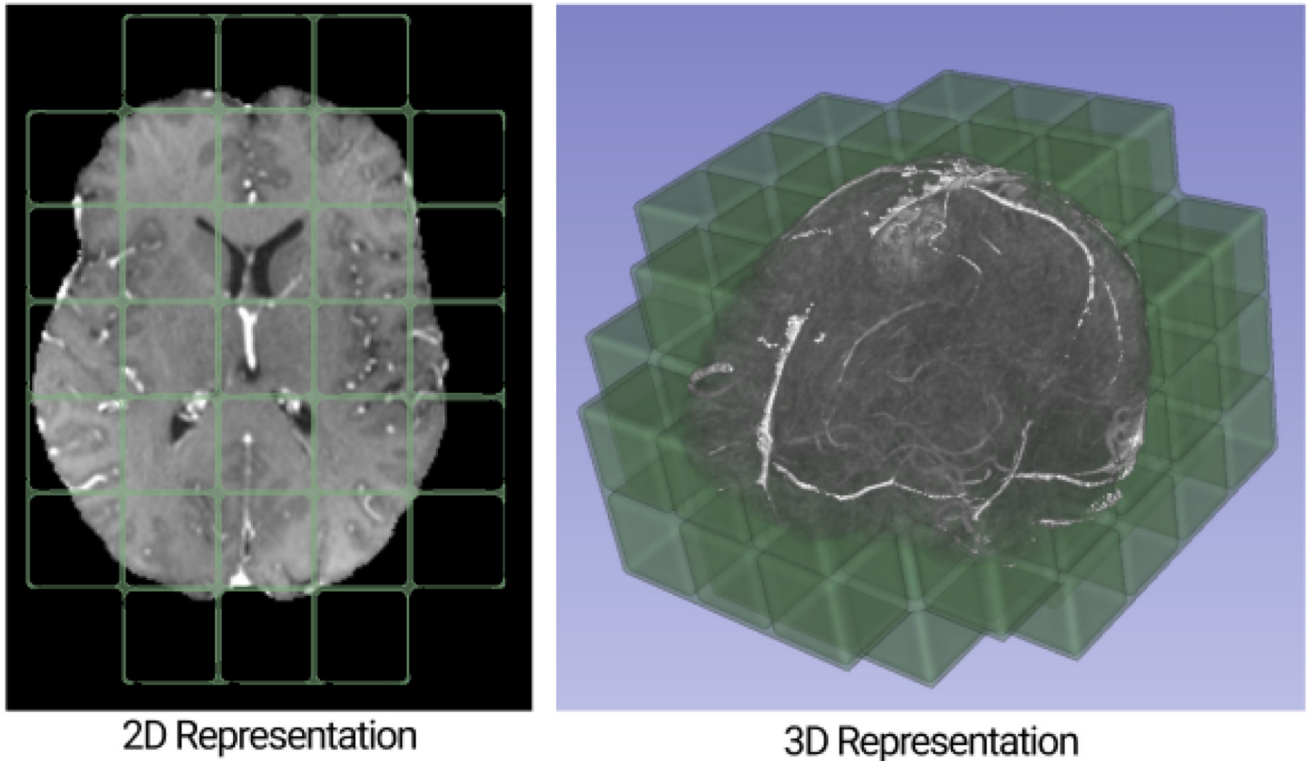


**Fig. 2.** An example of a typical preprocessing script for a neuroimaging pipeline. Data is converted from a directory of DICOM files into an internal representation in NumPy, corrected for intensity bias using 3DSlicer’s N4ITKBiasCorrection tool, co-registered across sequences using 3DSlicer’s BRAINSFit tool, and skull-stripped using a neural network trained in DeepNeuro.

Graphic created using the vector graphics program Figma. Visualization created via DeepNeuro and 3DSlicer. This figure and subsequent figures created by DeepNeuro code can be reproduced at <https://bit.ly/2TOFSEY>.

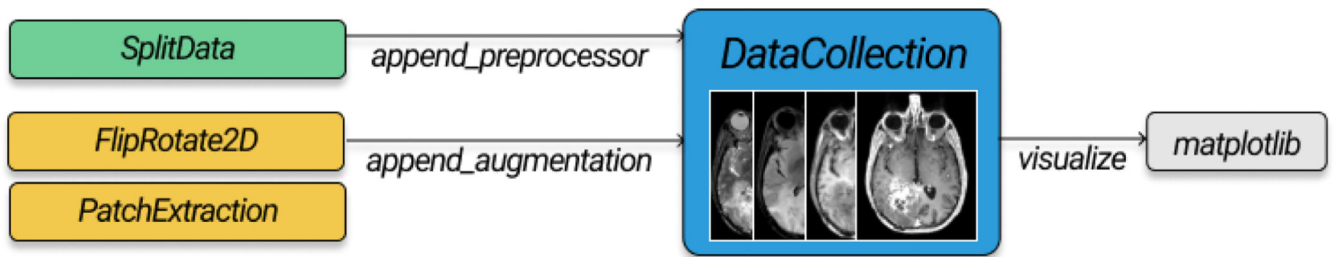


32x32x32 mm Patching Matrix from DeepNeuro's *ModelPatchesInference* Output Object.

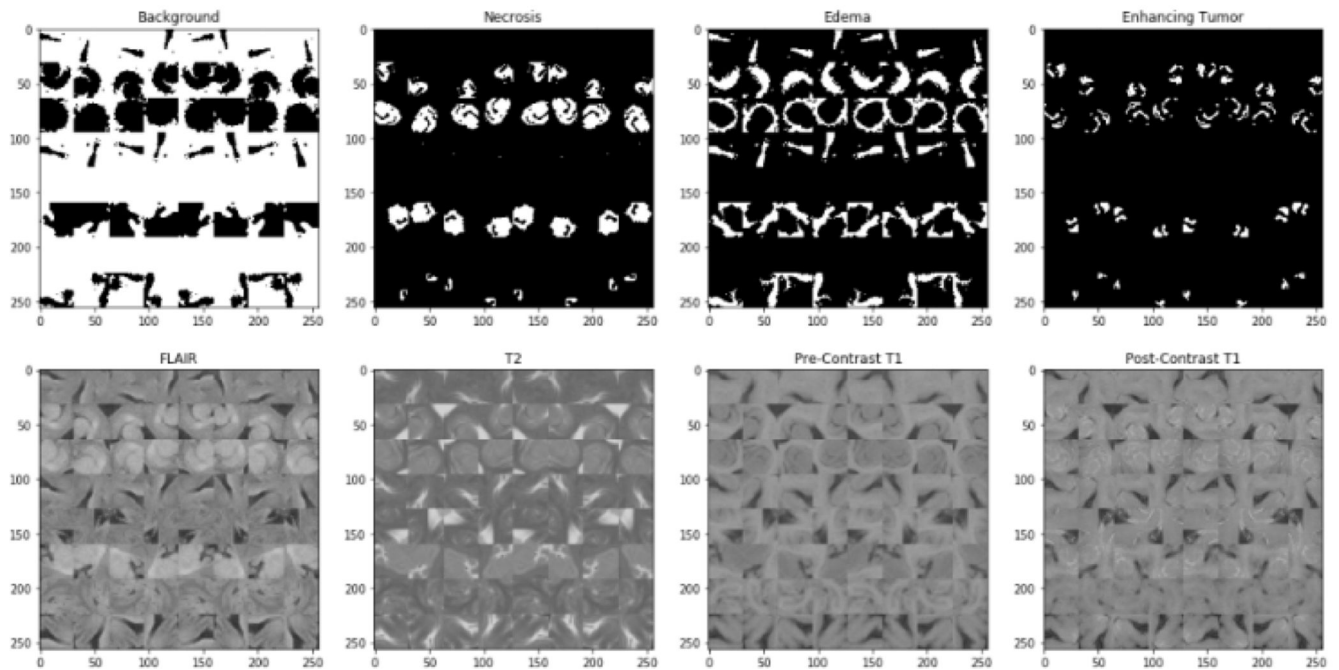


**Fig. 3.**

A schematic for how DeepNeuro divides images into patches using the *ModelPatchesInference* object. Using default settings, patches are not selected if they contain only zero values. Visualizations in this figure were generated by 3DSlicer, and the patch schematic grids can be recreated using the *PatchDiagram* object in DeepNeuro. Graphic created using the vector graphics program Figma. Visualization created via DeepNeuro and 3DSlicer.



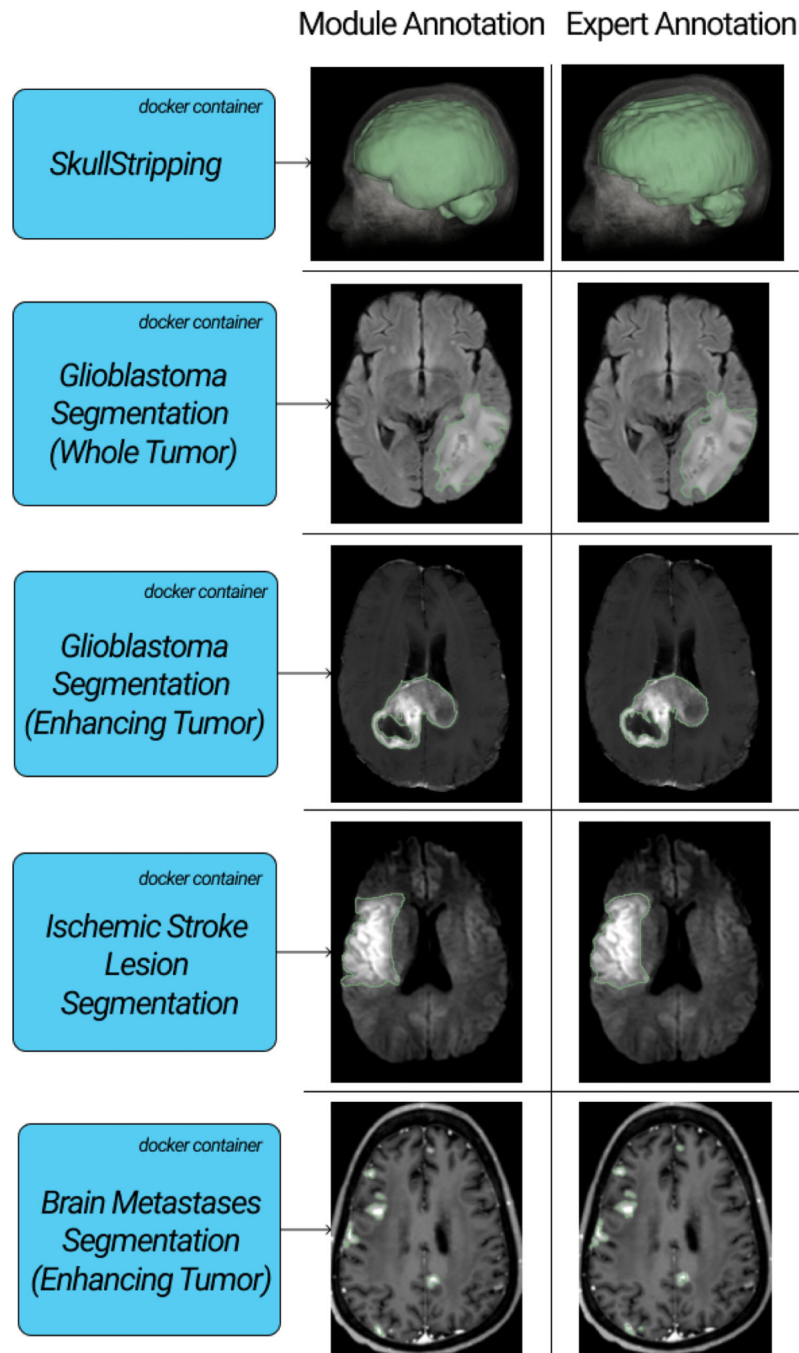
32x32x32 Patches and Ground Truth Labels Extracted from Glioblastoma Multi-Tissue Segmentation Data



**Fig. 4.**

A sample data preview generated by DeepNeuro’s *visualize* module for a glioblastoma segmentation use case. In this case, four MR sequences have been subdivided into patches, and have had flipping and 90 degree rotation augmentations applied to them in the axial dimension. In the generated graphic, labels for edema, enhancing tumor, necrosis, and non-pathological tissues (“background”) are also displayed for each image patch. This visualization can be reproduced in DeepNeuro’s “Loading, preprocessing, and Augmenting Data Using DeepNeuro” tutorial on Google Colaboratory.

Graphic created using the vector graphics program Figma. Visualization created via DeepNeuro and 3DSlicer.

**Fig. 5.**

A visualization of results compared with expert annotations for DeepNeuro's currently-available public modules. Expert annotations are performed by neurooncologists with 5+ years of experience in the case of the brain metastases and glioblastoma segmentations, a neuroradiologist with 9 years of experience in the case of the brain segmentations, and a neurology stroke fellow supervised by a stroke neurologist with 23 years of experience in the case of the stroke segmentations.

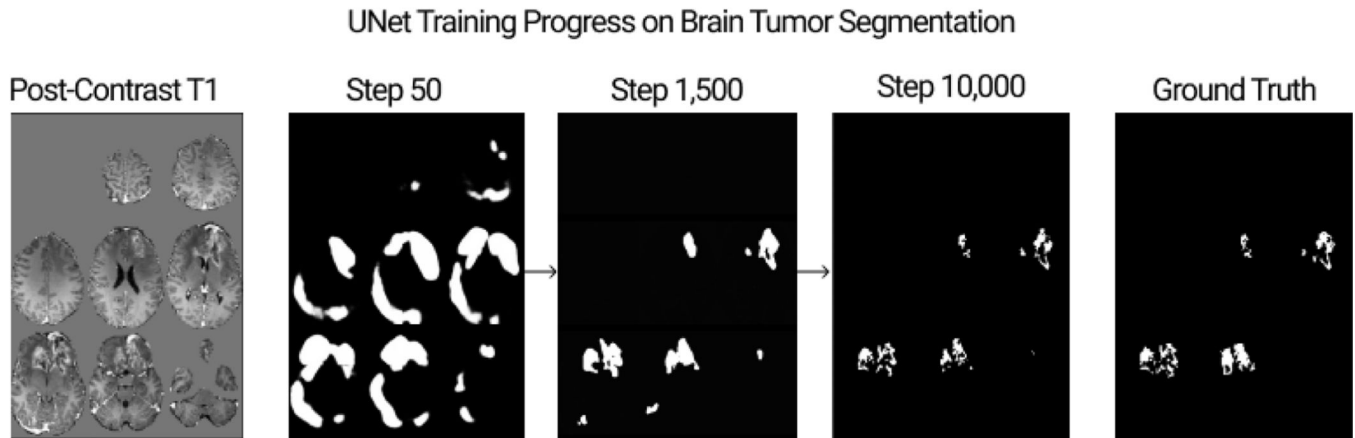
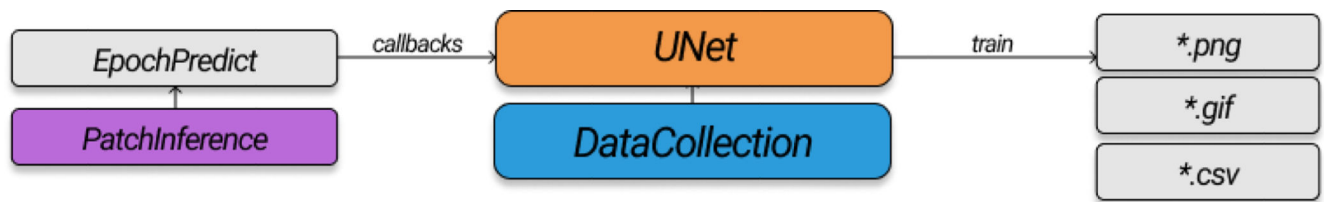
Graphic created using the vector graphics program Figma. Visualization created via DeepNeuro and 3DSlicer.

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript



**Fig. 6.**

A visualization and schematic of epoch-wise prediction in DeepNeuro. At user-defined intervals during model training, DeepNeuro can run lesion segmentations on a sample 3D volumetric image, and then save a mosaic image of slices from that image to disc. In this case, an algorithm is being trained to segment enhancing tumor from contrast-enhanced T1 MR imaging at three separate training steps, with test segmentations marked by white voxels. Early in the training, segmentations are characterized by false positives in non-pathological brain tissues; later, specificity is increased.

Graphic created using the vector graphics program Figma. Visualization created via DeepNeuro and 3DSlicer.