PocketNet: A Smaller Neural Network for Medical Image Analysis

Adrian Celaya, Jonas A. Actor, Rajarajesawari Muthusivarajan, Evan Gates, Caroline Chung, Dawid Schellingerhout, Beatrice Riviere, and David Fuentes

***Abstract*— Medical imaging deep learning models are often large and complex, requiring specialized hardware to train and evaluate these models. To address such issues, we propose the PocketNet paradigm to reduce the size of deep learning models by throttling the growth of the number of channels in convolutional neural networks. We demonstrate that, for a range of segmentation and classification tasks, PocketNet architectures produce results comparable to that of conventional neural networks while reducing the number of parameters by multiple orders of magnitude, using up to 90% less GPU memory, and speeding up training times by up to 40%, thereby allowing such models to be trained and deployed in resource-constrained settings.**

***Index Terms*— Neural network, segmentation, pattern recognition and classification.**

I. Introduction

DEEP learning is an increasingly common framework for automating and standardizing essential tasks related to medical image analysis that would otherwise be subject to wide variability. For example, delineating regions of interest (i.e., image segmentation) is necessary for computer-assisted diagnosis, intervention, and therapy [1]. Manual image segmentation is a tedious, time-consuming task whose results are often subject to wide variability among users [2], [3]. On the other hand, fully automated segmentation can substantially reduce the time required for target volume delineation and produce more consistent segmentation masks [2], [3]. Over the last several years, deep learning methods have produced impressive results for segmentation tasks such as labeling tumors or various anatomical structures [4], [5], [6], [7].

However, the performance of deep learning methods comes at an enormous computational and monetary cost, independent of concerns about data quality. Training networks to convergence can take several days or weeks using specialized computing equipment with sufficient computing capacity and available memory to handle large imaging datasets. The cost of a workstation with suitable hardware specifications for training large deep learning models ranges from roughly $5,700 to $49,000, whereas a dedicated deep learning-enabled server blade can range from $31,500 to $134,000 [8]. Cloud-based solutions offer a more economical option for training models by allowing users to pay for time on accelerated computing instances. However, the cost of such instances ranges from $3 to $32 per hour, and additional measures must be implemented to protect patient privacy [9]. The latter may involve an institution entering into a service agreement with a cloud-computing resource provider. The cost of such an agreement is another consideration that must be taken into account.

We wish to reduce the costs - in model size, training time, and memory requirements - associated with training deep learning models while preserving their performance. To do so, we propose the PocketNet paradigm for deep learning models, *a straightforward modification to existing architectures that substantially reduces the number of parameters and maintains the same performance as the original architecture*. This modification questions the long-held assumption that doubling the number of features after each downsampling operation (i.e., pooling or convolution) is necessary for convolutional neural networks [4], [5], [6], [7], [10], [11], [12], [13], [14], [15]. Our work demonstrates the effectiveness of our PocketNets in several 3D segmentation tasks and a classification problem. We also show that the PocketNet paradigm carries several practical benefits - less GPU memory required for training, less time required for training, and faster inference times.

Manuscript received 14 September 2022; accepted 20 November 2022. Date of publication 25 November 2022; date of current version 3 April 2023. This work was supported in part by the Tumor Measurement Initiative through the MD Anderson Strategic Research Initiative Development (STRIDE) and in part by the National Science Foundation under Award NSF-2111147 and Award NSF-2111459. The work of Adrian Celaya was supported by the Department of Defense through the National Defense Science and Engineering Graduate Fellowship Program. The work of Jonas A. Actor and Evan Gates was supported by the Training Fellowship from the Gulf Coast Consortia, on NLM Training Program in Biomedical Informatics and Data Science under Grant T15LM007093. The work of Dawid Schellingerhout and David Fuentes was supported in part by under Grant R21CA249373. The work of Beatrice Riviere was supported by under Grant NSF-DMS2111459. *(Adrian Celaya and Jonas A. Actor contributed equally to this work.) (Corresponding author: Adrian Celaya.)*

Adrian Celaya is with the Rice University, Houston, TX 77005 USA, and also with the University of Texas MD Anderson Cancer Center, Houston, TX 77030 USA (e-mail: aecelaya@rice.edu).

Jonas A. Actor was with Rice University, Houston, TX 77005 USA, and also with the University of Texas MD Anderson Cancer Center, Houston, TX 77030 USA. He is now with the Sandia National Laboratories, Albuquerque, NM 87123 USA (e-mail: jonasactor@gmail.com).

Rajarajesawari Muthusivarajan, Caroline Chung, and Dawid Schellingerhout are with the University of Texas MD Anderson Cancer Center, Houston, TX 77030 USA (e-mail: rmuthusivarajan@mdanderson.org; cchung3@mdanderson.org; dawid.schellingerhout@mdanderson.org).

Evan Gates was with the University of Texas MD Anderson Cancer Center, Houston, TX 77003 USA. He is now with the University of Washington, Seattle, WA 98195 USA (e-mail: egates1@uw.edu).

Beatrice Riviere is with Rice University, Houston, TX 77005 USA (e-mail: riviere@rice.edu).

David Fuentes is with the University of Texas MD Anderson Cancer Center, Houston, TX 77030 USA, and also with Rice University, Houston, TX 77005 USA (e-mail: dtfuentes@mdanderson.org).

Digital Object Identifier 10.1109/TMI.2022.3224873

A. Previous Work

The last several years have seen several attempts to decrease the number of parameters in deep learning architectures in medical imaging. Broadly, these attempts fall into two categories: post-processing tools and architecture design strategies. More specifically, pruning (post-processing tool), depth-wise separable (DS) convolutions (architecture design strategy), and filter reductions (architecture design strategy) are known to help mitigate the overparameterization of deep convolutional neural networks (CNNs) for 3D medical image segmentation [16], [17], [18]. These methods, alone and combined, give rise to many of the novel, efficient deep learning architectures that are currently popular. We briefly survey these network reduction strategies below.

Introduced by LeCun et al., the purpose of pruning is to remove the redundant connections within a neural network [16]. Pruning starts with a large pre-trained model and involves deleting weights and iteratively retraining the model until a significant drop in performance occurs. Pruning in medical image analysis reduces the required inference time and GPU memory while maintaining model performance [19], [20], [21]. However, network pruning is a post-processing step that is applied to an existing pre-trained network; although useful, pruning does not solve the demands of training large, overparameterized models, which requires a great deal of memory and high-end computing hardware.

Network architecture design strategies for reducing the number of parameters in deep neural networks for medical image analysis include DS convolutions and reduction of the number of feature maps at each layer. DS convolutions are well known for reducing the number of parameters associated with convolution [17], [22], [23]. DS convolution factorizes the standard convolution operation into two distinct steps: depth-wise convolution followed by a point-wise convolution. A normal convolution layer has a number of parameters that is quadratic in the number of channels. In contrast, DS convolutions have a number of parameters that is linear in the number of channels. In practice, recently developed medical neural network architectures take advantage of DS convolution to reduce the number of parameters in their networks by up to a factor of 5 while achieving results comparable with those of conventional convolution [24], [25]. However, 3D DS convolution is not supported in standard deep learning packages and requires more memory than standard convolution during training due to the storage of an extra gradient layer.

By convention, the number of feature maps doubles after each downsampling operation in a CNN. This growth in the number of feature maps accounts for a large portion of the number of training parameters. With this in mind, perhaps the simplest way to reduce overparameterization in a deep learning model is to reduce the number of feature maps. Van der Putten et al. explore this idea in [18] by dividing the number of feature maps in the decoder portion of a U-Net by a constant factor r. For increasing values of r, the number of parameters in the decoder is reduced by up to a factor of 100, and segmentation performance remains the same. However, this approach introduces another hyperparameter r that needs to be tuned, does not reduce the number of parameters in the encoder branch of the U-Net architecture, and is only applicable to segmentation models.

B. Novel Contributions

This paper makes the following novel contributions:

1) We propose the PocketNet paradigm. This paradigm builds from the definition of multigrid methods from numerical linear algebra. (Section II-A)
2) We demonstrate that PocketNet architectures perform comparably with their conventional neural network architecture counterparts. We measure PocketNet performance on three segmentation problems and one classification problem, using data from recent public medical imaging challenges. (Section III-A)
3) We profile the memory footprint for training conventional and PocketNet architectures, highlighting that PocketNet reduces the memory footprint for training models. During this profiling, we also track training time per epoch, showing that PocketNet models take less wall-time to train than do their conventional counterparts. (Section III-B)
4) We perform a controlled study of the effects of doubling the number of feature maps in convolutional neural networks, and additionally compare the intensities of the learned activations for voxel-wise classification (Sections III-C and III-D). To our knowledge, this constitutes the first controlled study of the effects of feature map doubling in convolutional deep learning architectures for medical imaging tasks.

II. Materials and Methods

A. The PocketNet Paradigm

We propose a modification to existing network architectures that dramatically reduces the number of parameters while also retaining performance. Many common network architectures for imaging tasks rely on manipulating images (or image features) at multiple scales because natural images encapsulate data at multiple resolutions. As a result, most CNN architectures – including many popular state-of-the-art methods such as nnUNet [14] and HRNet [13] – follow a pattern of downsampling and upsampling, following the intuition of the original U-Net paper [4] that popularized this approach. In the architecture first presented there, the number of feature maps (i.e., channels) in each convolution operator is doubled each time the resolution of the images decreases; the justification being that the increased number of feature maps offsets the loss of information from downsampling. This idea, of compensating for lost information by increasing the number of features, can be traced before U-Net, to the original ImageNet paper [11] and earlier. In all of these architectures, from ImageNet to U-Net to the state-of-the-art architectures today, the recurring refrain is that training is limited by compute availability due to the network's size, since the number of parameters in all of these architectures grows exponentially as the number of channels is doubled.

However, other classical methods that manipulate images (or other signals) at multiple scales assume a hierarchy of scales i.e. that information can be decomposed into coarse-scale and fine-scale features *independently*; most prominent among such methods are those based on wavelets [26] and on multigrid methods [27]. Intrinsic to these methods is the construction of a series of grids of appropriate resolution. At each resolution, the constructed grid allows for the specific frequencies to be resolved, without aliasing. Because of this hierarchy of scales, and that specific sets of frequencies are resolved by specific grids at specific scales, the information capacity of coarser scales is guaranteed to be less than that of finer scales, and as a result, fewer operations (for multigrid solvers) and memory (for wavelets) are required: information "lost" by downsampling into a smaller, coarser subspace is accounted for at a grid of different resolution, and when images are downsampled to a coarser resolution, it is not necessary to double the number of channels or dimensions to preserve the information capacity at each downsampling instance. As a result, we propose that the doubling of the channels at each resolution in CNN architectures like U-Net is not intrinsically necessary, since each depth in these architectures corresponds to features at a different scale.

A generic U-Net framework is fully written out in Algorithm 1.

Algorithm 1 U-Net

Input: Tensor u_D, integers D, ν

procedure BLOCK(ν, u) ▷ Tensor u, integer ν
  for $i = 1 : \nu$ **do**
    $u \leftarrow \sigma\,(K_i * u \ + b_i)$ ▷ Convolution block
  return u

procedure U-NET(u_D, D, ν)
  $u_D \leftarrow \texttt{Block}(\nu, u_D)$ ▷ Initial computation

  Encoder
  for $d = D - 1$ to 1 **do**
    $u_d \leftarrow \Pi_{d+1}^{d} u_{d+1}$ ▷ Downsample
    $u_d \leftarrow \texttt{Block}(\nu, u_d)$

  Coarsest resolution
  $v_0 \leftarrow \Pi_1^0 u_1$
  $v_0 \leftarrow \texttt{Block}(\nu, v_0)$

  Decoder
  for $d = 1$ to D **do**
    $v_d \leftarrow \Pi_{d-1}^{d} v_{d-1}$ ▷ Upsample
    $v_d \leftarrow u_d + v_d$ ▷ Skip connection
    $v_d \leftarrow \texttt{Block}(\nu, v_d)$
  return $\phi(K_{\text{out}} * v_D + b_{\text{out}})$

In the `Block` procedure of this algorithm, each convolution kernel K_i has some number of channels-in and channels-out that depend on the layer's depth in the 'U' of the architecture. The overall depth D used in this architecture is commonly set to $D = 3$ to $D = 6$ [4], [5]. If the convolutions have c_{in} channels-in and c_{out} channels-out at the network's finest resolution, the convolutions in the next layer will have $2c_{\text{in}}$ channels-in and $2c_{\text{out}}$ channels-out. Subsequently, at resolution depth d, there are $2^d c_{\text{in}}$ channels-in and $2^d c_{\text{out}}$ channels-out for the convolutions. As a result, the number of parameters in a CNN grows exponentially with increasing depth, and this exponential growth is the driving factor for the large size of image segmentation networks.

We remark that Algorithm 1 is nearly identical to a single V-cycle of a geometric multigrid solver for solving the linear system of equations $Au = f$, where the linear system A and the unknown variable u relate to a geometric grid involving multiple resolutions [27], [28]. An algorithm for a V-cycle is shown in Algorithm 2.

Algorithm 2 Multigrid V-Cycle, Adapted From [28]

Input: Matrix A_D, vectors f_D, u_D, integers D, ν

procedure BLOCK(ν, A, u, f) ▷ Matrix A, vectors u, f
  for $i = 1 : \nu$ **do**
    $u \leftarrow D^{-1}(f - (A - D)u)$ ▷ Iterative step
  return u

procedure V-CYCLE(A_D, u_D, f_D, D, ν)
  $u_D \leftarrow \texttt{Block}(\nu, A_D, u_D, f_D)$ ▷ Initial computation
  $r_D \leftarrow f_D - A_D u_D$ ▷ Residual update

  Encoder
  for $d = D - 1$ to 1 **do**
    $f_d, A_d \leftarrow \Pi_{d+1}^{d} r_{d+1}, A_{d+1}$ ▷ Downsample
    $u_d \leftarrow \texttt{Smooth}(\nu, A_d, 0, f_d)$
    $r_d \leftarrow f_d - A_d u_d$ ▷ Residual update

  Coarsest resolution
  $f_0, A_0 = \Pi_1^0 r_1, A_1$
  $v_0 = A_0^{-1} f_0$ ▷ Direct solve

  Decoder
  for $d = 1$ to D **do**
    $v_d \leftarrow \Pi_{d-1}^{d} v_{d-1}$ ▷ Upsample
    $v_d \leftarrow u_d + v_d$ ▷ Skip connection
    $v_d \leftarrow \texttt{Block}(\nu, A_d, u_d, f_d)$
  return v_D

The PocketNet paradigm, defined in Definition 2.1, exploits this similarity between multigrid methods and U-Net-like architectures. This paradigm proposes that the number of feature maps used at the finest resolution is sufficiently rich to capture the relevant information for the imaging task at hand and that doubling the number of channels is unnecessary. Instead of doubling the number of feature maps at every level of a CNN, we keep them constant, substantially reducing the number of parameters in our models in the process.

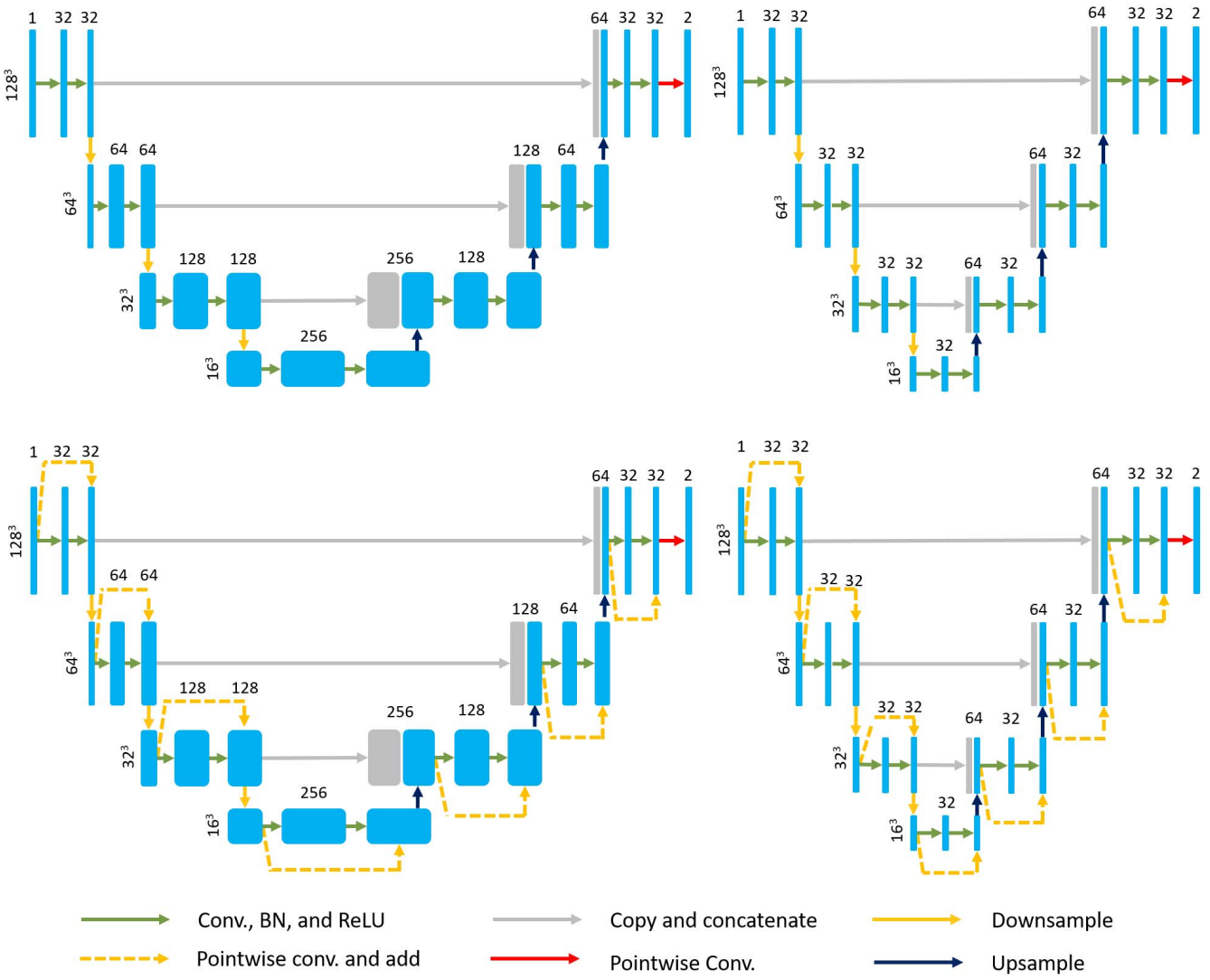


Fig. 1. (Left) Full U-Net (top) and ResNet (bottom) segmentation architectures where for each downsampling step, the number of feature maps doubles after each convolution layer. (Right) Pocket U-Net (top) and ResNet (bottom) segmentation architecture. In this PocketNet architecture, the number of feature maps resulting from each convolution layer remains constant regardless of spatial resolution, resulting in substantially fewer parameters overall.

We designate network architectures that keep the number of feature maps constant as *Pocket Networks*, or *PocketNets* for short, in the sense that these networks are "small enough to fit in one's back pocket". For example, "Pocket U-Net" refers to a U-Net architecture where we apply our proposed modification. Figure 1 provides a visual representation of the PocketNet paradigm applied to a U-Net. We present a formalized definition of the PocketNet paradigm in Definition 2.1.

Definition 2.1: A network architecture obeys the *PocketNet paradigm* if the range of all convolution operators (except the final output layer) present in the network is a subset of $\mathbb{R}^{h\times w\times c_{\text{out}}}$, where c_{out} is fixed. Such a network is called a Pocket Network, or PocketNet for short.

While the relationship between multigrid algorithms and U-Nets has been explored (see [29] and references therein), the PocketNet paradigm is applicable to any CNN architecture, regardless of whether or not the architecture is similar in overall structure to a U-Net.

The number of parameters saved in the PocketNet architectures is substantial. Assume that a PocketNet and its corresponding full-sized architecture are identical apart from the doubling of the number of channels in standard convolutions. For simplicity's sake, we further assume that the number of convolutions performed at each resolution is the same, denoted by C. Also for simplicity's sake, we assume that the window of each convolution kernel is isotropic, with a stencil width of k in each dimension. We denote the maximum resolution depth, i.e. the number of downsampling operations in the network architecture, as D. In a full-sized network, a convolution kernel at resolution depth d operating on an nD image (for $n = 2$ or $n = 3$) with a stencil width k has $k^n \left(c_{\text{in}}2^d\right)\left(c_{\text{out}}2^d\right)$ parameters. Therefore, the full-sized network has the following

number of parameters

$$n_{\text{full}} = \sum_{d=0}^{D} Ck^n(c_{\text{in}}2^d)(c_{\text{out}}2^d) = \frac{1}{3}Ck^n c_{\text{in}}c_{\text{out}}\left(4^{D+1}-1\right). \tag{1}$$

On the other hand, the convolution kernels in a PocketNet architecture have $k^n c_{\text{in}}c_{\text{out}}$ parameters at each resolution depth, resulting in the following number of parameters

$$n_{\text{pocket}} = \sum_{d=0}^{D} Ck^n c_{\text{in}}c_{\text{out}} = (D+1)Ck^n c_{\text{in}}c_{\text{out}}. \tag{2}$$

Therefore, the PocketNet paradigm reduces the number of parameters in a network by up to a factor of

$$\text{savings} = \frac{n_{\text{full}}}{n_{\text{pocket}}} = \frac{4^{D+1}-1}{3(D+1)}) \approx \frac{4^D}{D}. \tag{3}$$

This analysis highlights that the growth of parameters with increasing depth is exponential for full networks but only linear for PocketNets.

B. Experiments

1) Data: We test PocketNet on a range of recent public medical imaging challenge datasets. These datasets encompass three segmentation problems and one classification task, all with various data set sizes, complexity, and modalities. Two of our segmentation tasks - binary liver segmentation in the MICCAI Liver and Tumor Segmentation (LiTS) Challenge 2017 dataset [30] and single-contrast brain extraction in the Neurofeedback Skull-stripped (NFBS) repository [31] - are comparatively simple. We use these datasets for baseline comparisons, much like the MNIST Fashion and CIFAR-10 datasets are used to evaluate newly proposed classification architectures. The third segmentation task - multilabel tumor segmentation in the MICCAI Brain Tumor Segmentation (BraTS) Challenge 2020 dataset [32], [33], [34] - is commonly viewed as a more complex and technical segmentation problem than LiTS or NFBS segmentation. Therefore, we use BraTS data for our performance benchmarks (see Section III-B). Finally, our classification task - binary classification in the COVIDx8B dataset [35] - evaluates the PocketNet paradigm with a much larger dataset, demonstrating its appropriateness for pertinent problems other than image segmentation. These datasets and their related pre-processing and post-processing methods are described below.

a) LiTS Data: For the LiTS dataset, we perform binary liver segmentation. This dataset consists of the 131 CT scans from the MICCAI 2017 Challenge's multi-institutional training set. These scans vary significantly in the number of slices in the axial direction and voxel resolution, although all axial slices are at 512×512 resolution. As a result, we use the preprocessing steps proposed by nnUNet to handle this variability [14]. We resample each image to the median resolution of the training data in the x and y-directions and use the 90th percentile resolution in the z-direction. For intensity normalization, we window each image according to the foreground voxels' 0.5 and 99.5 percentile intensity values across all of the training data. This scheme results in windowing from -17 to 201 HU. We also apply z-score normalization according to the foreground voxels' mean and standard deviation. The LiTS dataset is available for download https://competitions.codalab.org/competitions/17094#learn_the_details-overview.

b) NFBS Data: The segmentation task for the NFBS dataset is extraction (i.e., segmentation) of the brain from MR data. The NFBS dataset consists of 125 T1-weighted MR images with manually labeled ground truth masks. All images are provided with an isotropic voxel resolution of $1\times1\times1$ mm^3 and are of $256 \times 256 \times 192$ resolution. For pre-processing, we apply z-score intensity normalization. The NFBS dataset is available for download at http://preprocessed-connectomes-project.org/NFB_skullstripped/.

c) BraTS Data: The BraTS training set contains 369 multimodal scans from 19 institutions. Each set of scans includes a T1-weighted, post-contrast T1-weighted, T2-weighted, and T2 Fluid Attenuated Inversion Recovery volume and a multilabel ground truth segmentation. We merge the labels in each ground truth segmentation and perform whole tumor segmentation for our analysis. All volumes are provided at an isotropic voxel resolution of $1 \times 1 \times 1$ mm^3, co-registered to one another, and skull stripped, with a size of $240 \times 240 \times 155$. We crop each image according to the brainmask (i.e., non-zero voxels) and apply z-score intensity normalization on only non-zero voxels for pre-processing. The BraTS training dataset is available for download at https://www.med.upenn.edu/cbica/brats2020/registration.html.

d) COVIDx8B Data: The classification task for the COVIDx8B dataset is COVID-19 detection on 2D chest x-rays. The COVIDx8B dataset consists of a training set with 15,952 images and an independent test set with 400 images. The training set is class-imbalanced, with 13,794 COVID-19 negative images and 2,158 COVID-19 positive images. However, the COVIDx8B test set is class-balanced, with 200 COVID-19–negative and 200 COVID-19–positive images. We resize each image to a resolution of 256×256 and apply z-score intensity normalization as pre-processing steps. The COVIDx8B dataset is available for download at https://github.com/lindawangg/COVID-Net/blob/master/docs/COVIDx.md.

2) Network Architectures: For each of our tasks and datasets, we compare the performance of various full-sized architectures with their PocketNet counterparts.

a) Segmentation Architectures: We examine the effects of our proposed modification strategy using five segmentation architectures - U-Net [5], ResNet [7], DenseNet [6], HRNet [13], and nnUNet [14]. The U-Net, ResNet, and DenseNet architectures are similar. They use the same U-Net backbone with variations in their convolution blocks (see Figure 2) but are highly prevalent architectures in the literature for 3D medical image segmentation [15]. HRNet is not as prevalent in the literature as U-Net and its variants but does represent a fundamentally different deep learning architecture for 3D segmentation. Rather than using a "U" shape, where we continually coarsen (i.e., downsample via pooling)

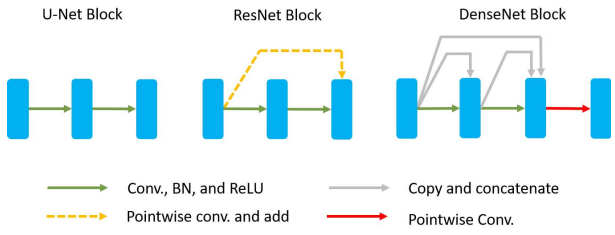


Fig. 2. Block designs for U-Net (left), ResNet (center), and DenseNet (right).

high-resolution features, HRNet preserves features at each resolution level. The nnUNet architecture provides a state-of-the-art baseline for our analysis.

b) COVIDx8B architectures: Our classification architecture for the COVIDx8B dataset is a U-Net encoder with four downsampling layers. The final layer of the U-Net encoder is flattened via global max-pooling and passed to a fully connected layer for the network's final output. Visually, this architecture is represented by the left half of the Pocket U-Net architecture in Figure 1.

3) Training Protocols and Hyperparameters: For each task (e.g., segmentation and classification), we initialize the first layer in each network with 32 feature maps and use the Adam optimizer [36]. The learning rate is set to 0.0003. Training for all segmentation tasks uses a batch size of two and a batch size of 32 for COVIDx8B classification. For all segmentation tasks, we use a patch size of $128 \times 128 \times 128$ and apply the same random augmentation described in [14]. To evaluate a predicted segmentation mask's validity, we use the Sorensen-Dice Similarity Coefficient (Dice), the 95$^{\text{th}}$ percentile Hausdorff distance, and the average surface distance. Implementations of these metrics are available through the SimpleITK Python package [37], [38], [39]. For the segmentation tasks, our loss function is calculated as an L_2 relaxation of the Dice score; for a true segmentation Y_{true} and a predicted segmentation Y_{pred}, our L_2-Dice loss function is taken from [40] and is given as

$$Loss_{Dice}(Y_{true}, Y_{pred}) = \frac{\left\| Y_{true} - Y_{pred} \right\|_2^2}{\|Y_{true}\|_2^2 + \left\| Y_{pred} \right\|_2^2}. \quad (4)$$

For the nnUNet architecture, we add binary cross entropy to the Dice loss shown above.

For the classification tasks, we use categorical cross-entropy as our loss function, with outputs being of two classes (COVID-19 positive and negative). We use the receiver operating characteristic area under the curve (AUC) metric to evaluate each classification model's validity. This metric is available via the scikit-learn Python package. Our models are implemented in Python using TensorFlow (v2.8.0) and trained on an NVIDIA Quadro RTX 8000 GPU [41]. All network weights are initialized using the default initializers from TensorFlow. All other hyperparameters are left at their default values. The code for each network architecture is available at github.com/aecelaya/pocketnet.

4) Inference and Post-Processing: We perform inference on test images using a sliding window approach for each segmentation task, where the window size equals the patch size set during training. After each window prediction, we slide the window by half the size of the patch. Additionally, we apply a Gaussian importance weighting ($\sigma = 0.125$) to each window prediction [14].

For post-processing, we try each of the following strategies for each segmentation task and network:

1) Apply morphological clean-up - i.e., erode by two voxels, retain the largest connected component, dilate by one voxel, and fill holes.
2) Only retains the largest connected component.
3) Do nothing.

We select the strategy that yields the highest mean Dice as our final post-processing strategy for a given task and network.

We note that the inference and post-processing steps described above are applied to *all* segmentation tasks for every network we test.

III. Results

A. Comparable Accuracy

We use the training parameters described in Section II-B.3 to train the architectures listed in Section II-B.2. For the segmentation tasks described in Section II-B.1, we employ a five-fold cross-validation scheme to obtain predictions for each dataset and architecture. For classification on the COVIDx8B dataset, we train each model with the training set and generate predictions on the test set. These results of these experiments are shown in Tables I and II.

For the BraTS and LiTS datasets, we generally do not see significant performance differences ($p < 0.05$ [Wilcoxon signed-rank test]) between the full and pocket architectures. In the cases where we see a significant difference in performance, the differences are small, with median Dice scores differing by less than 1% and median Hausdorff distances differing by less than a fraction of a millimeter. We also do not see a clear pattern in significant performance differences between each pocket and full architecture. Given this and the fact that these differences are generally small, we believe that the outperformance of a full-sized network by its pocket counterpart or vice versa can be explained by the stochastic nature of training each network.

For the NFBS dataset, we generally see statistically significant differences in performance. However, we again note that these differences are small, and there is no clear pattern for which architecture (i.e., full or pocket) outperforms. These differences are so small (less than 0.0001 for Dice) that they bear no clinical significance. Such minor differences, in this case, are imperceptible and meaningless in a practical setting.

For all three segmentation tasks, these insignificant or minor differences in performance indicate a reduction in the number of parameters by more than an order of magnitude.

B. Performance Analysis

Using the training parameters described in Section II-B.3, we profile the training performance of a full U-Net and

TABLE I

ACCURACY OF DEEP LEARNING MODELS ON A SET OF MEDICAL IMAGING TASKS FOR POCKET VS. FULL-SIZED ARCHITECTURES. THE POCKETNET MODELS' ACCURACY SCORES ARE GENERALLY COMPARABLE (WITHIN 1% OR LESS) OF THE FULL MODELS' ACCURACY SCORES. ACCURACY SCORES FOR SEGMENTATION TASKS (LITS, NFBS, BRATS) ARE EVALUATED USING DICE SIMILARITY COEFFICIENT, HAUSDORFF 95 DISTANCE, AND AVERAGE SURFACE DISTANCE

Task	Architecture	Variant	# Params (M)	Dice			Hausdorff 95 (mm)			Avg. Surface (mm)		
				Mean (Std)	Median	p-value	Mean (Std)	Median	p-value	Mean (Std)	Median	p-value
LiTS	U-Net	Full	90.5	0.922 (0.098)	0.952	NS	9.715 (23.95)	0.967	NS	2.713 (3.637)	1.446	NS
		Pocket	0.8	0.930 (0.059)	0.952		7.845 (21.67)	0.959		2.343 (2.832)	1.384	
	ResNet	Full	91.9	0.932 (0.097)	0.958	NS	9.105 (28.40)	0.729	NS	2.528 (4.876)	1.185	NS
		Pocket	0.8	0.928 (0.104)	0.956		10.38 (26.90)	0.742		2.615 (3.915)	1.262	
	DenseNet	Full	135.9	0.933 (0.056)	0.955	NS	18.97 (50.65)	0.781	NS	3.478 (7.297)	1.234	*
		Pocket	1.3	0.930 (0.078)	0.951		10.16 (27.21)	1.000		2.588 (3.463)	1.405	
	HRNet	Full	6.5	0.939 (0.067)	0.958	*	6.126 (19.37)	0.756	NS	2.082 (2.889)	1.202	NS
		Pocket	0.7	0.937 (0.055)	0.954		8.805 (26.97)	0.787		2.455 (3.931)	1.304	
	nnUNet	Full	31.2	0.933 (0.092)	0.953	*	6.187 (19.09)	0.800	NS	2.300 (4.030)	1.339	*
		Pocket	0.8	0.944 (0.039)	0.956		2.844 (6.285)	0.785		1.738 (1.432)	1.263	
BraTS	U-Net	Full	90.5	0.889 (0.090)	0.915	NS	5.627 (13.59)	1.414	NS	1.695 (2.242)	1.081	*
		Pocket	0.8	0.891 (0.088)	0.914		5.089 (12.18)	1.414		1.750 (3.375)	1.023	
	ResNet	Full	91.9	0.902 (0.083)	0.927	NS	4.648 (11.59)	1.000	NS	1.469 (2.012)	0.842	NS
		Pocket	0.8	0.904 (0.077)	0.927		4.007 (10.34)	1.000		1.308 (1.543)	0.834	
	DenseNet	Full	135.9	0.887 (0.106)	0.917	*	5.833 (12.97)	1.414	NS	1.693 (2.560)	0.938	NS
		Pocket	1.3	0.889 (0.103)	0.917		5.935 (12.98)	1.414		1.730 (2.792)	0.929	
	HRNet	Full	6.5	0.900 (0.077)	0.926	*	5.729 (13.95)	1.000	NS	1.659 (2.315)	0.927	NS
		Pocket	0.7	0.897 (0.083)	0.922		5.202 (12.37)	1.000		1.523 (2.042)	0.900	
	nnUNet	Full	31.2	0.907 (0.076)	0.929	NS	3.472 (8.733)	1.000	NS	1.150 (1.484)	0.750	NS
		Pocket	0.8	0.908 (0.070)	0.931		3.586 (10.57)	1.000		1.194 (2.241)	0.749	
NFBS	U-Net	Full	90.5	0.988 (0.002)	0.988	**	0.000 (0.000)	0.000	NS	0.419 (0.124)	0.384	***
		Pocket	0.8	0.988 (0.002)	0.987		0.000 (0.000)	0.000		0.451 (0.133)	0.423	
	ResNet	Full	91.9	0.988 (0.002)	0.989	***	0.000 (0.000)	0.000	NS	0.329 (0.090)	0.313	***
		Pocket	0.8	0.988 (0.002)	0.989		0.000 (0.000)	0.000		0.405 (0.124)	0.392	
	DenseNet	Full	135.9	0.988 (0.002)	0.989	**	0.000 (0.000)	0.000	NS	0.441 (0.159)	0.401	NS
		Pocket	1.3	0.988 (0.002)	0.989		0.000 (0.000)	0.000		0.446 (0.166)	0.401	
	HRNet	Full	6.5	0.988 (0.002)	0.989	***	0.000 (0.000)	0.000	NS	0.412 (0.141)	0.373	NS
		Pocket	0.7	0.988 (0.002)	0.989		0.000 (0.000)	0.000		0.411 (0.137)	0.380	
	nnUNet	Full	31.2	0.989 (0.002)	0.990	***	0.000 (0.000)	0.000	NS	0.272 (0.049)	0.260	***
		Pocket	0.8	0.989 (0.002)	0.989		0.000 (0.000)	0.000		0.284 (0.045)	0.276	

NS, p-value > 0.05
*, p-value < 0.05
**, p-value < 0.01
***, p-value < 0.001

TABLE II

ACCURACY OF DEEP LEARNING MODELS ON THE COVIDX CLASSIFICATION TASK FOR POCKET VS. FULL-SIZED ARCHITECTURES. THE POCKETNET MODELS' ACCURACY SCORES ARE GENERALLY COMPARABLE (WITHIN 1% OR LESS) TO THE FULL MODELS' ACCURACY SCORES. ACCURACY SCORES FOR THE COVIDX CLASSIFICATION TASK ARE EVALUATED USING AUC, AND HENCE DO NOT HAVE STANDARD DEVIATIONS OR P-VALUES

Task	Architecture	Variant	# Parameters (M)	AUC
COVIDx	U-Net Encoder	Full	1.2	0.997
	U-Net Encoder	Pocket	0.02	0.999
	State-of-the-art [35]	Full	8.8	0.994

a Pocket U-Net using the BraTS dataset. Namely, we measure peak GPU memory utilization during training and the average time per training step for varying batch sizes for each network using the TensorFlow Profiler [42]. To ensure accurate comparisons of performance, we conduct these experiments on a Google Colaboratory notebook with a dedicated NVIDIA Tesla T4 GPU with 16 GB of available memory.

The GPU memory usage and training time per step for this experiment are shown in Figure 3; we see that our PocketNet architecture reduces memory usage and speeds up training time for every batch size. Specifically, the Pocket U-Net reduces the peak memory usage for training by between 28.3% and 87.7%, with smaller batch sizes resulting in greater savings. This relationship is possibly due to the increasing portion of GPU memory allocated for storing data as the batch size increases. The PocketNet models improve the average time per training step by between 25.0% and 43.2%, with larger batches yielding greater time savings. This behavior may be due to larger batch sizes taking advantage of the computational parallelism of modern GPUs.

In addition to training performance, we benchmark the inference speed of full-sized networks and their PocketNet counterparts. We choose throughput in terms of the number of images predicted per second as our metric for inference speed. Throughput is a popular way of benchmarking the inference speed of a network [43], [44], [45], [46]. Higher throughput means that a network can perform faster inference in a given computing environment. We measure the throughput of the full and pocket variants for several networks and compute the percent speed up of the PocketNet for several different image sizes. The percent speed up is given by the following:

$$\%\ \text{speed up} = 100 \times \frac{\text{pocket throughput} - \text{full throughput}}{\text{full throughput}}$$

Table III shows the inference throughput of each architecture for various image sizes. For architectures with a standard U-Net backbone (i.e., U-Net, ResNet, and DenseNet), we see modest improvements in inference speed for the PocketNet

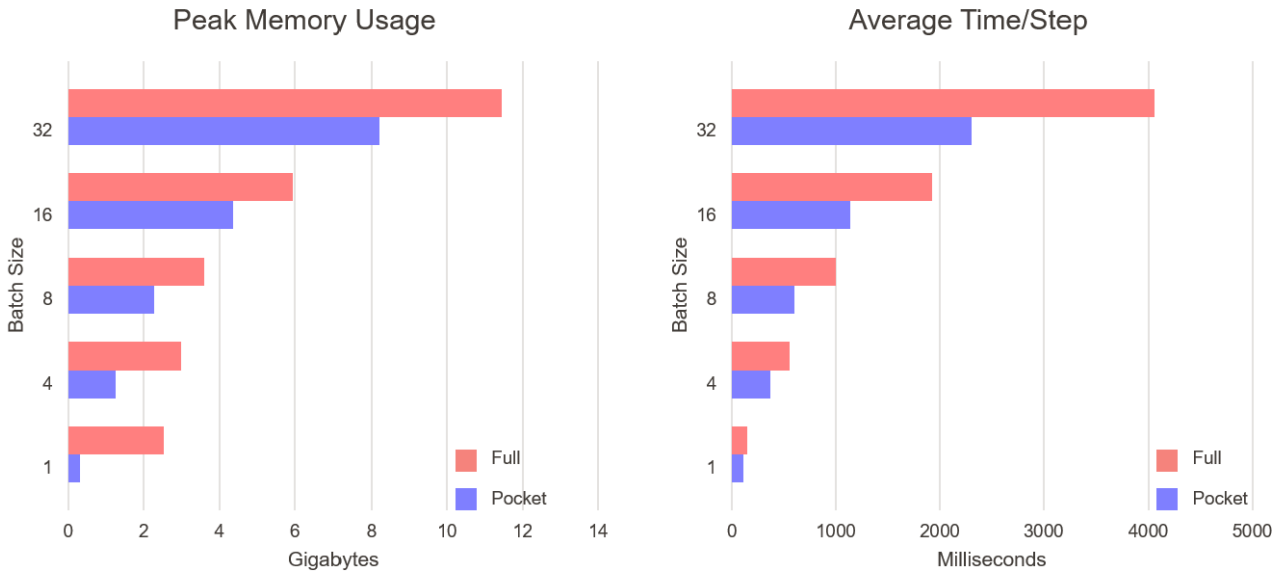


Fig. 3. (Left) Peak memory usage during training on the BraTS dataset for Pocket U-Net vs Full U-Net for varying batch sizes. The PocketNet architecture results in memory savings of between 28.3% and 87.7%. (Right) Average time per training step on BraTS dataset for Pocket U-Net vs Full U-Net for varying batch sizes. The PocketNet models speed up the average time per training step by between 25.0% and 43.2%.

TABLE III
INFERENCE THROUGHPUT FOR POCKETNET VS. FULL ARCHITECTURES OVER MULTIPLE IMAGE SIZES USING A SINGLE NVIDIA RTX 8000 GPU. IN THIS CASE, THROUGHPUT MEASURES THE NUMBER OF IMAGES PREDICTED PER SECOND. HIGHER THROUGHPUT IMPLIES FASTER INFERENCE SPEED. IN EVERY CASE, WE OBSERVE FASTER INFERENCE TIMES WITH POCKET MODELS, WITH SPEED-UPS (SEE FORMULA ABOVE) VARYING FROM 1 TO 17%

Image Size	Architecture	Variant	Throughput (image/sec)	% Speed Up
32×32×32	U-Net	Full	17.86	5.89
		Pocket	18.98	
	ResNet	Full	16.67	1.17
		Pocket	16.86	
	DenseNet	Full	14.37	3.74
		Pocket	14.93	
	HRNet	Full	13.16	12.24
		Pocket	14.99	
64×64×64	U-Net	Full	15.50	2.48
		Pocket	15.90	
	ResNet	Full	12.94	12.54
		Pocket	14.79	
	DenseNet	Full	12.61	11.60
		Pocket	14.27	
	HRNet	Full	10.53	16.53
		Pocket	12.61	
128×128×128	U-Net	Full	7.84	5.88
		Pocket	8.33	
	ResNet	Full	6.72	3.70
		Pocket	6.98	
	DenseNet	Full	5.38	7.86
		Pocket	5.84	
	HRNet	Full	3.89	13.41
		Pocket	4.49	

variants that range from 1% to 12%. A possible explanation for these minor improvements in inference speed is the highly parallelized computation of convolution operators on modern GPUs. For HRNet, we see more significant improvements in inference throughput ranging from 12% to 17%. Within the HRNet architecture, we see more non-convolution operations like upsampling, downsampling, and addition than in U-shaped architectures, which may explain the more significant increases in throughput for the Pocket HRNet architecture.

C. Ablation Study

To assess the effects of feature map doubling in U-shaped architectures, we perform an ablation study on a standard U-net using the LiTS dataset. In the first iteration of the ablation study, we start with a Full U-Net where we double the number of feature maps at every resolution level. In the next iteration, we construct a U-Net where we stop doubling feature maps after the second-to-last resolution level (i.e., $8 \times 8 \times 8$ at $d = 1$). We continue until we arrive at the Pocket U-Net. By performing this ablation study, we can determine at what resolution does feature map doubling become important for the network's accuracy.

For each of these networks, we perform a five-fold cross-validation using training and inference parameters described in Sections II-B.3 and II-B.4. Table IV shows the results of each iteration. In every case, we see small differences in the distribution of the resulting Dice scores. This small difference in performance among iterations suggests that doubling the number of feature maps at each resolution level might be unnecessary.

D. Feature Activation Analysis

We conjecture that the comparable performance between the PocketNet and full architectures is due to both networks having similar representation capabilities, that ultimately both networks build similar representations of the image data as they compute the final segmentations. To test whether the networks learn similar features, we look at the mean of the

TABLE IV
ACCURACY OF U-NET ARCHITECTURES ON LITS DATASET WHERE FEATURE MAP DOUBLING STOPS AFTER A GIVEN DEPTH *d*. IN EVERY CASE, WE SEE SMALL DIFFERENCES IN THE DISTRIBUTION OF THE RESULTING DICE SCORES

Stop feature map doubling after depth d	Max features per convolution	# Parameters (M)	Dice	
			Mean $\pm$ Std.	Median
$d = 0$ i.e. Full U-Net	1024	90.5	0.922 $\pm$ 0.098	0.952
$d = 1$	512	60.0	0.921 $\pm$ 0.109	0.953
$d = 2$	256	24.3	0.923 $\pm$ 0.069	0.952
$d = 3$	128	8.40	0.907 $\pm$ 0.145	0.950
$d = 4$	64	2.60	0.922 $\pm$ 0.105	0.951
$d = 5$ i.e. PocketNet	32	0.80	0.930 $\pm$ 0.059	0.952

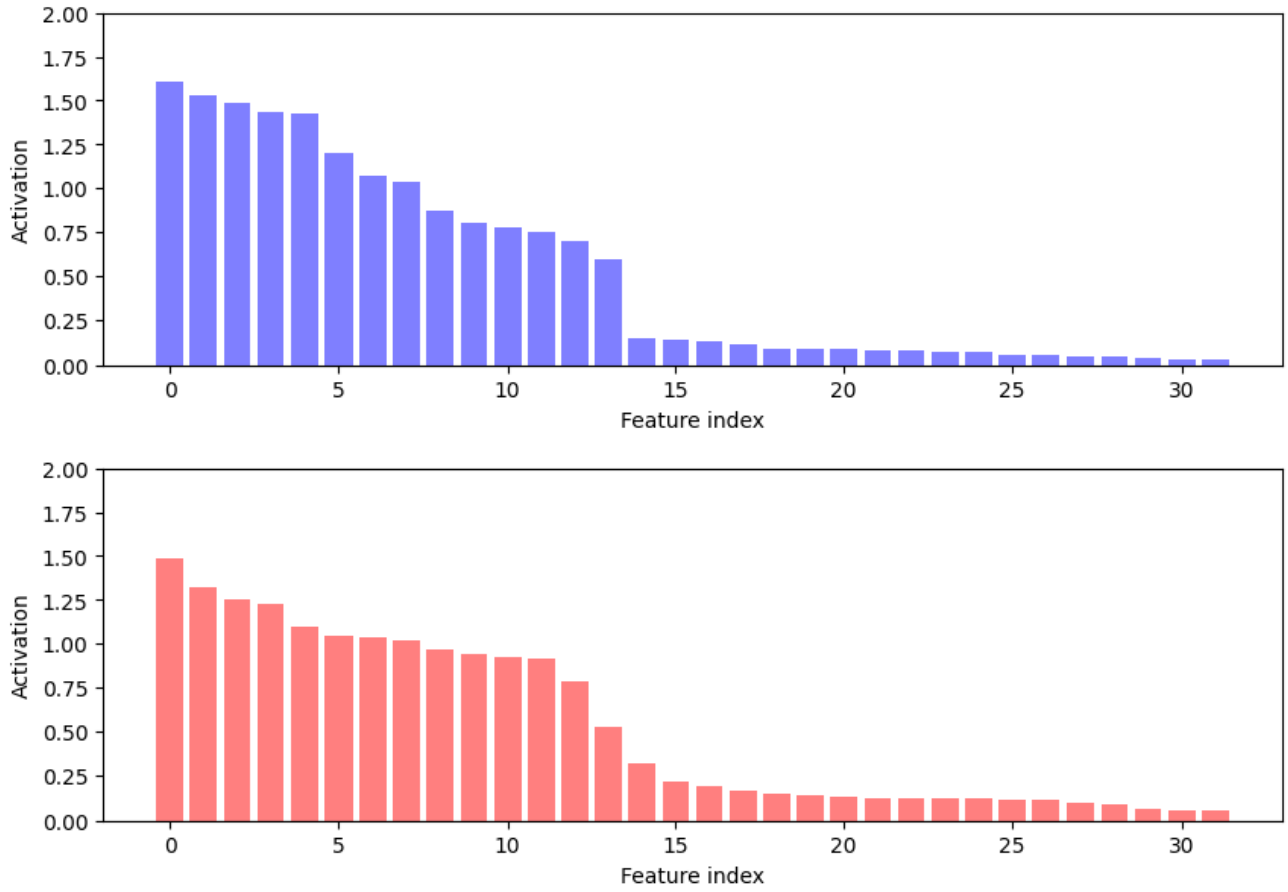
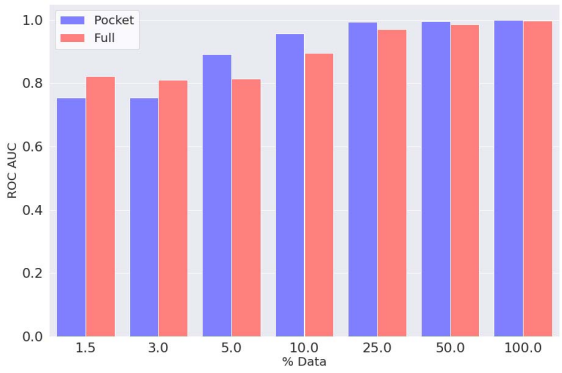


Fig. 4. Feature map activations in final layer before voxel-wise classification in a Pocket U-Net (top) and full U-Net (bottom) for the BraTS dataset. In both cases, we see a similar number of features being activated with roughly the same intensities.

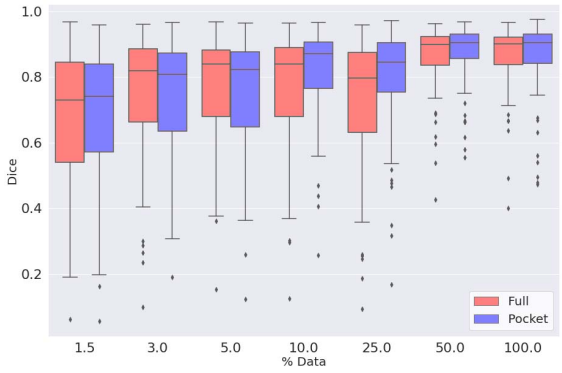
feature map activations in the final layer before voxel-wise classification in full and Pocket U-Nets trained on BraTS data. In this way, we can determine if both networks are capturing similar features and information. For each image in the test set, we examine the output of the activation functions from the final layer before classification and measure the size of the response towards the convolution feature, by taking the channel-wise average of the resulting feature maps of the activation output. This process is repeated for the entire test set. More precisely, let F_i be the average of the i^{th} feature map resulting from activation outputs from the final layer before classification over the entire test set. Let f_i^j be the i^{th} feature map resulting from the activation functions for test patch j. For each $i = 1, \ldots, 32$, we have

$$F_i = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \frac{1}{V^j} \sum_{k=1}^{V^j} \left(f_i^j\right)_k,$$

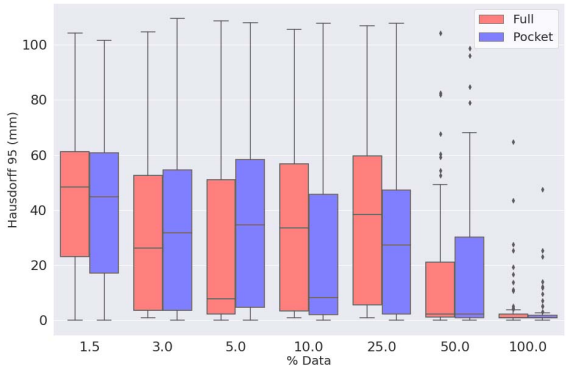
where N_{test} is the number patches in the test set, V^j is the volume of test patch j, and $\left(f_i^j\right)_k$ is the intensity of the k^{th} voxel in f_i^j. Figure 4 shows the averages of the resulting feature maps. We see a similar number of features being activated with similar intensities for both cases. This similarity suggests that the full and Pocket U-Nets learn similar latent feature



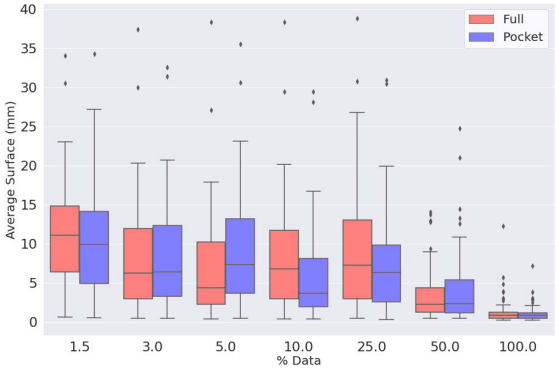
(a) Classification performance on COVIDx8B data for PocketNet vs. Standard U-Net encoder classifier, trained using various subsets of the training data.



(b) Distributions of Dice scores on BraTS test set for a full U-Net and a pocket U-Net for varying training set sizes.



(c) Distributions of Hausdorff 95 distances on BraTS test set for a full U-Net and a pocket U-Net for varying training set sizes.



(d) Distributions of average surface distances on BraTS test set for a full U-Net and a pocket U-Net for varying training set sizes.

Fig. 5. Testing results from using small subsets of data for PocketNet vs. full architectures on COVID19×8B classification and BraTS segmentation challenges.

representations used for the final voxel-wise classification. Note that we sort the mean feature activations from highest to lowest for visual purposes. This order does not matter because the indexing in any hidden layer can always be permuted by the next layer.

E. Model Saturation

A possible concern is that PocketNet models, due to their reduced parameter count, could saturate earlier during training than do full-sized architectures, which could result in the comparable performance we observe in our results in Section III-A. To test this, we repeat the experiments described above for the COVIDx8B and BraTS data challenges using successively less data in the training set. For every iteration, we keep a fixed validation and test set. For the COVIDx8B dataset, we fix 10% of the training data as a validation set and use the original test set. Similarly, for the BraTS data, we take 20% of the training data as a test set and use 10% of the remaining patients as a validation set. Additionally, we do not use data augmentation for this particular experiment. The results of this are shown in Figure 5a and Figure 5b.

In Figure 5a, we see that the AUC values increase for both the PocketNet and full architectures as the size of the training set increases. Furthermore, we observe that these AUC values plateau at 1.0 (i.e., perfect prediction), and the PocketNet classifier saturates sooner than its full-sized counterpart. These observations suggest that the reduced architecture resulting from the PocketNet paradigm learns faster with fewer data points than its full-sized counterpart. Similarly, Figures 5b, 5c, and 5d show that the segmentation accuracy of the full and pocket U-Net architectures improves as the training set size increases. Both architecture types show the expected improvement in performance with each increase in the dataset size, plateauing to similar distributions.

IV. Discussion

Our results show that large numbers of parameters (millions or tens of millions) may not be necessary for deep learning in medical image analysis, as comparable performance is achievable with substantially smaller networks using the same architectures but without doubling the number of channels

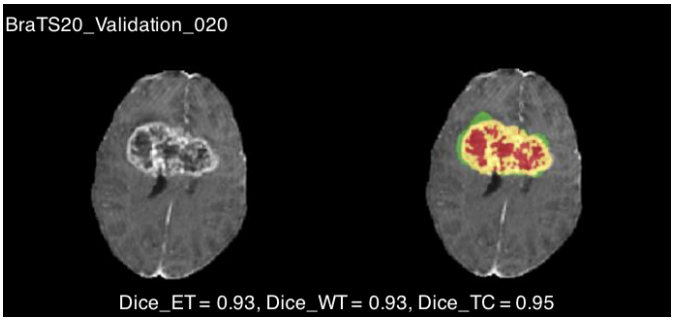


Fig. 6. Multi-class Pocket DenseNet segmentation on BraTS 2020 Validation image. Enhancing Tumor (ET), Whole Tumor (WT), and Tumor Core (TC) Dice scores are in line with state-of-the-art predictions.

at coarser resolutions. This suggests that overparameterization, which is increasingly regarded as a key reason why neural networks learn efficiently, might not be as critical as previously suggested [47], [48], [49]. However, we note that our PocketNets may still be are overparameterized, and the combination of our proposed PocketNet paradigm with other model reduction techniques should be explored. For example, replacing the traditional convolution layers with DS convolution layers in our Pocket ResNet for LiTS liver segmentation further reduces the number of parameters to roughly 10,000. Pruning an already trained PocketNet model may also potentially yield further parameter reductions.

The deep learning tasks presented in this study are all single-label segmentation or binary classification. The goal of ongoing and future work using the PocketNet paradigm is to test this approach on more complex domains such as BraTS multi-class tumor segmentation and LiTS tumor segmentation. Figure 6 shows an example of a multi-class segmentation prediction mask produced by a Pocket DenseNet. Our results for PocketNet architectures applied to the BraTS multi-class segmentation task are available at https://www.cbica.upenn.edu/BraTS20/lboardValidation.html under the team name "aecmda" and will be updated periodically.

Regarding Section III-E and the results shown in Figure 5, we note that the results for each task using 1.5% of the data look surprisingly good. For the segmentation task (i.e., BraTS tumor segmentation), a closer analysis reveals that while the full and pocket U-Nets can identify brain tumors in the BraTS dataset with a small percentage of the available training data, they tend to produce rather noisy predictions. Figure 7 illustrates an example of these predictions. In both cases, the networks erroneously predict brighter areas of the MR images as tumors. We hypothesize that the images in the BraTS dataset are similar enough that even with a small percentage of available data, each network can learn to predict rough segmentations of the target. This behavior, however, supports our claim that the pocket and full-sized U-Nets behave similarly with varying dataset sizes. Again, Figure 7 illustrates this by showing an example prediction from each network for small and large training sets. In both cases, we see similar behavior, with both architectures producing noisy predictions with fewer data and more accurate predictions as we increase the dataset size.

Additionally, the Dice coefficient may be too forgiving of a metric for this type of analysis. Even for noisy predictions, we can achieve Dice scores of about 0.7. To mitigate this, we compute the Hausdorff 95 and average surface distances for each set of predictions in Figures 5c and 5d, respectively. With the Hausdorff 95 metric, we see that predictions from training on small datasets generally result in significant errors with substantial variation. However, for larger datasets, we see that for each network variant, the distribution of Hausdorff 95 distances converges to smaller values with reduced variance. We again see similar behavior with the average surface distance. We see significant errors with considerable variance for each network variant for smaller datasets. These distributions plateau to smaller values with less variance with increased training set size for both full and pocket networks. These cases support our initial claim that both networks exhibit similar behavior with different dataset sizes.

For the classification results in Figure 5, we again argue that the images in the dataset are similar enough to each other that even with a small percentage taken as a training set, each architecture can roughly discriminate between the two classes.

When we employ PocketNet models, we achieve similar performance to full-sized networks while enjoying the advantages of faster training times and lower memory requirements. The smaller models produced by our proposed PocketNet paradigm can potentially lower the entry costs (computational and monetary) of training deep learning models in resource-constrained environments without access to specialized computing equipment. With less GPU memory required for training, cheaper hardware can be purchased, or less expensive cloud computing instances can be used to train deep learning models for medical image analysis. The faster training times for PocketNets can also reduce costs by reducing the number of hours spent training models on cloud computing instances.

As to *why* the PocketNet models perform at least comparable to their counterpart full models, the similarity in intensities of each signal in the final layer activation maps suggests that the models ultimately learn the same representations. Despite the reduced number of parameters, the approximation space that the Pocket architectures can represent is comparable to the approximation space that the full models can achieve. This conjecture is supported by the ablation study results in Table IV: the median Dice scores for e.g. U-Net is nearly identical, regardless of the depth that the feature map stops doubling. In this study, the additional features supplied at depths where the doubling continued did not improve the model's performance, as the approximation spaces are all similar regardless of whether the number of features was doubled at that depth. This ablation study suggests that, since there is no increased benefit of having larger models with the number of features doubling per layer, that for these medical imaging problems, doubling the number of channels is unnecessary and PocketNet models can be used to achieve comparable accuracy instead. Since these smaller models are just as expressive (and capable) as their full counterparts, these models can be trained

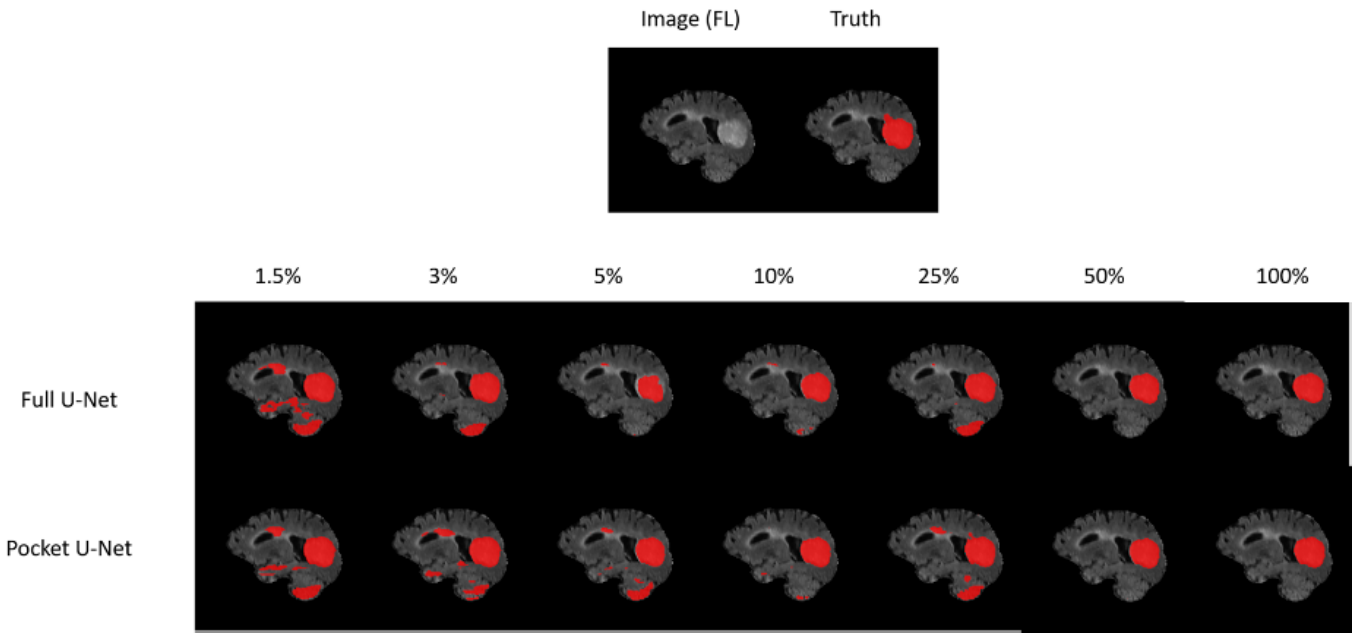


Fig. 7. Example predictions for full and pocket U-Net on BraTS dataset with varying percentages of the dataset used for training. In both cases, we see similar behavior, with both architectures producing noisy predictions with fewer data and more accurate predictions as we increase the dataset size.

(and later, deployed) with cheaper hardware or by provisioning smaller cloud instances, saving time, money, and effort by institutions performing deep learning medical image analysis.

REFERENCES

[1] N. Sharma and L. M. Aggarwal, "Automated medical image segmentation techniques," *J. Med. Phys.*, vol. 35, no. 1, pp. 3–14, Jan. 2010.

[2] D. Thomson et al., "Evaluation of an automatic segmentation algorithm for definition of head and neck organs at risk," *Radiat. Oncol.*, vol. 9, no. 1, pp. 1–12, Dec. 2014.

[3] E. Ermis et al., "Fully automated brain resection cavity delineation for radiation target volume definition in glioblastoma patients using deep learning," *Radiat. Oncol.*, vol. 15, no. 1, pp. 1–10, Dec. 2020.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.* Cham, Switzerland: Springer, 2015, pp. 234–241.

[5] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-Net: Learning dense volumetric segmentation from sparse annotation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.* Cham, Switzerland: Springer, 2016, pp. 424–432.

[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 630–645.

[8] *GPU Cloud, Workstations, Servers, Laptops for Deep Learning*. [Online]. Available: https://lambdalabs.com/

[9] *Amazon AWS EC2 Pricing*. [Online]. Available: https://aws.amazon.com/ec2/pricing/

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 84–90.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[13] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep high-resolution representation learning for human pose estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5693–5703.

[14] F. Isensee, P. F. Jaeger, S. A. A. Kohl, J. Petersen, and K. H. Maier-Hein, "nnU-Net: A self-configuring method for deep learning-based biomedical image segmentation," *Nature Methods*, vol. 18, no. 2, pp. 203–211, Dec. 2020.

[15] N. Siddique, S. Paheding, C. P. Elkin, and V. Devabhaktuni, "U-Net and its variants for medical image segmentation: A review of theory and applications," *IEEE Access*, vol. 9, pp. 82031–82057, 2021.

[16] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2, 1989, pp. 598–605.

[17] R. Ye, F. Liu, and L. Zhang, "3D depthwise convolution: Reducing model parameters in 3D vision tasks," in *Advances in Artificial Intelligence*. Springer, 2019, pp. 186–199.

[18] J. Van Der Putten, F. Van Der Sommen, and P. H. N. De With, "Influence of decoder size for binary segmentation tasks in medical imaging," in *Proc. SPIE*, vol. 11313, 2020, pp. 276–281.

[19] Z. Zhou et al., "UNet++: A nested U-Net architecture for medical image segmentation," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer, 2018, pp. 3–11.

[20] Y. Weng, T. Zhou, Y. Li, and X. Qiu, "NAS-Unet: Neural architecture search for medical image segmentation," *IEEE Access*, vol. 7, pp. 44247–44257, 2019.

[21] A. Feng-Ping and L. Zhi-Wen, "Medical image segmentation algorithm based on feedback mechanism convolutional neural network," *Biomed. Signal Process. Control*, vol. 53, Aug. 2019, Art. no. 101589.

[22] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.

[23] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[24] N. Alalwan, A. Abozeid, A. A. ElHabshy, and A. Alzahrani, "Efficient 3D deep learning model for medical image semantic segmentation," *Alexandria Eng. J.*, vol. 60, no. 1, pp. 1231–1239, Feb. 2021.

[25] K. Qi et al., "X-Net: Brain stroke lesion segmentation based on depthwise separable convolution and long-range dependencies," in *Medical Image Computing and Computer Assisted Intervention—MICCAI*. Cham, Switzerland: Springer, 2019, pp. 247–255.

[26] T. F. Chan and J. J. Shen, *Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods*, vol. 94. Philadelphia, PA, USA: SIAM, 2005.

[27] J. H. Bramble, *Multigrid Methods*. Evanston, IL, USA: Routledge, 2018.
[28] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: SIAM, 2003.
[29] J. He and J. Xu, "MgNet: A unified framework of multigrid and convolutional neural network," *Sci. China Math.*, vol. 62, no. 7, pp. 1331–1354, Jul. 2019.
[30] P. Bilic et al., "The liver tumor segmentation benchmark (LiTS)," 2019, *arXiv:1901.04056*.
[31] B. Puccio, J. P. Pooley, J. S. Pellman, E. C. Taverna, and R. C. Craddock, "The preprocessed connectomes project repository of manually corrected skull-stripped T1-weighted anatomical MRI data," *GigaScience*, vol. 5, no. 1, p. 45, Dec. 2016.
[32] B. H. Menze et al., "The multimodal brain tumor image segmentation benchmark (BRATS)," *IEEE Trans. Med. Imag.*, vol. 34, no. 10, pp. 1993–2024, Oct. 2015.
[33] S. Bakas et al., "Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge," 2018, *arXiv:1811.02629*.
[34] S. Bakas et al., "Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features," *Sci. Data*, vol. 4, no. 1, Dec. 2017, Art. no. 170117.
[35] L. Wang, Z. Q. Lin, and A. Wong, "COVID-Net: A tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images," *Sci. Rep.*, vol. 10, p. 19549, Nov. 2020.
[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[37] B. C. Lowekamp, D. T. Chen, L. Ibáñez, and D. Blezek, "The design of SimpleITK," *Front. Neuroinform.*, vol. 7, p. 45, Dec. 2013.
[38] Z. Yaniv, B. C. Lowekamp, H. J. Johnson, and R. Beare, "SimpleITK image-analysis notebooks: A collaborative environment for education and reproducible research," *J. Digit. Imag.*, vol. 31, no. 3, pp. 290–303, Jun. 2018.
[39] R. Beare, B. Lowekamp, and Z. Yaniv, "Image segmentation, registration and characterization in R with SimpleITK," *J. Stat. Softw.*, vol. 86, no. 8, pp. 1–35, 2018.
[40] J. A. Actor, D. T. Fuentes, and B. Rivière, "Identification of kernels in a convolutional neural network: Connections between the level set equation and deep learning for image segmentation," in *Proc. SPIE*, vol. 11313, 2020, Art. no. 1131317.
[41] F. Chollet et al., "Keras," Software available from Keras.org, Tech. Rep., 2015.
[42] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," Software available from tensorflow.org, Tech. Rep., 2015.
[43] K. Rungsuptaweekoon, V. Visoottiviseth, and R. Takano, "Evaluating the power efficiency of deep learning inference on embedded GPU systems," in *Proc. 2nd Int. Conf. Inf. Technol. (INCIT)*, Nov. 2017, pp. 1–5.
[44] R. Xu, F. Han, and Q. Ta, "Deep learning at scale on NVIDIA V100 accelerators," in *Proc. IEEE/ACM Perform. Model., Benchmarking Simul. High Perform. Comput. Syst. (PMBS)*, Nov. 2018, pp. 23–32.
[45] S. Mittal, "A survey on optimized implementation of deep learning models on the NVIDIA Jetson platform," *J. Syst. Archit.*, vol. 97, pp. 428–442, Jan. 2019.
[46] L. Mai, A. Koliousis, G. Li, A.-O. Brabete, and P. Pietzuch, "Taming hyper-parameters in deep learning systems," *ACM SIGOPS Operating Syst. Rev.*, vol. 53, no. 1, pp. 52–58, Jul. 2019.
[47] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 242–252.
[48] S. Arora, N. Cohen, and E. Hazan, "On the optimization of deep networks: Implicit acceleration by overparameterization," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 244–253.
[49] L. Rice, E. Wong, and Z. Kolter, "Overfitting in adversarially robust deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 8093–8104.