Quantifying Robustness of Medical Image Segmentation Networks Using TensorStars

Meghan Stuart
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, United States of America
mstuart@cs.unc.edu

Parasara Sridhar Duggirala
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, United States of America
psd@cs.unc.edu

***Abstract*—This paper presents a framework for quantifying robustness of U-Nets, a widely used neural network architecture in 3 dimensional biomedical image segmentation. We first present a metric for measuring robustness of biomedical image segmentation methods. We then propose TensorStars, a new representation to model the perturbation of a given medical image and overapproximate the effect of these perturbations on the image segmentation performed by a 3 dimensional U-Net. Given the large dimensionality of medical images, naive implementation of overapproximation results in poor scalability of robustness verification. We propose two main improvements, (1) leveraging hardware acceleration from GPUs for performing tensor operations, and (2) leverage the CPU parallelism for accelerating the ReLU overapproximation. We observe that these two enhancements improve the runtime by 10x when implemented on a 32 core CPU with NVidia GeForce RTX with 8GB VRam. We perform robustness analysis on a U-Net that performs segmentation of kidneys with cross entropy loss of less than 0.01. Our evaluation demonstrates that perturbing less than 3% of input voxels can result in misclassification of up to 20% of voxels in the segmentation output. To the best of authors knowledge, this is the first work that extends the robustness verification of neural networks to biomedical imaging data.**

I. Introduction

Inspired by the success of deep learning in various computer vision tasks such as classification and segmentation, researchers have extended these deep learning techniques to medical image analysis tasks such as classification, segmentation, and registration. Currently, neural networks are the state-of-the-art in performing most of the medical image analysis tasks. These neural networks are currently integrated into various workflow for treatment. For example, deep learning is frequently used and studied in the radiotherapy treatment workflow, to perform automatic segmentation of at-risk organs [1], [2], [3], for registration of images taken during multi-day treatments [4], [5], [6], for image enhancement like multi-modality image fusion [7], [8] or metal artifact reduction [9], [10], lesion detection and characterization [11], [12], landmark detection for fiducial-free motion tracking [13], and even medical image synthesis [14], [15] which must be used carefully to ensure proper model generalizability [16]. Given the safety critical nature of such integrating such neural networks into clinical procedures, one has to ensure that the outcomes of the neural networks deployed in biomedical images are robust. In order for deep learning technology to be integrated into the clinical workflow, clinicians must have confidence in the tool. They expect model transparency and interpretability, and require assurances of the model's ability to perform consistently through rigorous quality assurance testing during the initial implementation of the technology, as well as on an ongoing basis to ensure its adaptability to evolving imaging techniques and treatment protocols [17].

Empirical evaluation of neural networks in medical imaging and beyond, does not inspire such confidence. It has been shown that deep learning models, in general, lack generalizability to out of distribution samples [18]. In the specific domain of medical imaging, it has been shown in [19] that a model trained for pneumonia detection on lung radiographs on data from a specific hospital may have significantly poorer performance on the same diagnostic task using data from a different hospital. This may be due to differences in imaging technology, image processing or diagnostic protocols. *To provide confidence and explainability of neural networks used in medical imaging, in this paper, we verify the robustness of U-Nets, a widely used neural network architecture in segmentation. That is, given a set of perturbations (possibly uncountable), does the segmentation outcome of the U-Net change as a result of the perturbations.*

Unique Aspects of Biomedical Image Segmentation and its effects on Robustness Verification: There are significant differences between deep learning techniques used in biomedical image analysis when compared to disciplines such as computer vision or natural language processing. First, biomedial images are typically 3 dimensional in nature and it requires significant effort to generate high quality labeled data for segmentation and other downstream tasks. Second, the scale of data is significant smaller, that is, datasets often contain 30-50 patient data and data augmentation techniques play a key role in generating labeled data. Finally, the widely used architecture for 3 dimensional data, namely U-Nets have a unique feature called skipped-connection layers that are not commonly used in other disciplines. These differences require a careful examination of formulating and verifying robustness properties of biomedial image segmentation networks.

A lot of prior work on neural network verification was targeted towards demonstrating robustness of specific neural networks such as ACASXu [20] or VGG-16 [21]. However, in contrast, we do not verify a standard neural network available

https://doi.org/10.34727/2025/isbn.978-3-85448-084-6_11

on a public domain as that is not the modality of usage of neural networks in medical imaging domain. In fact, researchers in medical image analysis discourage using the same network for different data sets as each network is trained on data from a small number of patients.

Therefore, to assess the robustness of a biomedial image segmentation networks we propose the following workflow. First, we pick a widely used neural network architecture and training algorithm in the literature and train a new neural network on a publicly available data set. Next, we formulate important robustness properties over biomedial images and verify these properties on the trained neural network.

The authors understand the circularity in this argument. If the neural network is poorly chosen, then it would be unsurprising to find violations of robustness property. Our answer to this circularity is two-fold. (1) the architecture, training algorithm, and loss functions picked by us are widely accepted to be standard in the medical imaging literature and are now part of several commercial products, and (2) our main contribution is to not to demonstrate that a specific network violates or satisfies the properties, but rather the algorithmic procedure can indeed verify the properties of neural networks used in medical image analysis.

Another key difference between neural networks for segmentation and classification is the scale of the output. A segmentation network is required to assign a class for each pixel in the image, as opposed to a classification network that assigns a single class to the entire image. In the case of biomedical image segmentation, each voxel is classified to be either an *organ* or a *background* voxel. Therefore, verifying robustness of a segmentation network on a 32x32x32 biomedical image is equivalent to verifying the robustness of 32^3 or 32,768 individual classifications. In this paper, instead of constructing individual verification instances, we construct an overapproximation of the classification results on all the voxels and efficiently verify the robustness of the segmentation network.

Contributions of Our Paper: In this paper we propose a new representation called TensorStar for modeling perturbations of a given medical image. We show that noise models specific to medical imaging and noise models widely used in neural network verification can be modeled as TensorStars We then propose algorithms for constructing a conservative overapproximation of the segmentation output of a U-Net when given inputs in the TensorStar. We demonstrate that U-Nets for segmenting Kidneys on a given data set are not robust with respect to perturbations. Our verification demonstrates that while minor perturbations of the image does not lead to any change in segmentation outcome, changing merely 3% of voxels can change the segmentation of Kidney by 20%. **To the best of authors knowledge, this is the first paper on robustness verification for neural networks in medical imaging domain.**

II. Problem Formulation

We consider 3D images consisting of voxels. Each voxel is associated with a specific intensity recorded from the sensor. We model the space of images as $\mathbb{R}^{n_1 \times n_2 \times n_3}$ where n_1, n_2, and n_3 define the resolution of the image in x, y, and z dimensions. Typically the intensity of an image lies in the interval $[0, 255]$.

In this particular paper, we focus on segmentation networks. For the sake of simplicity, we focus on single object segmentation. That is, there is only one organ of interest in the image, and the segmentation network identifies whether a given voxel belongs to the organ or the background. Given an image I is an element of $\mathbb{R}^{n_1 \times n_2 \times n_3}$, a segmentation network $\mathcal{N} : \mathbb{R}^{n_1 \times n_2 \times n_3} \rightarrow \mathbb{B}^{n_1 \times n_2 \times n_3}$ where each voxel is tagged with a boolean value. A voxel is labelled as 1 if it is classified to be a part of the segment (organ) and 0 if it is classified to be part of the background. In general, medical images have same resolution in the two axial directions ($n_1 = n_2$). In research models, the images are often sampled such that the dimensions are the same in all three dimensions, i.e., $n_1 = n_2 = n_3$.

Our goal is to verify the robustness property of segmentation networks. That is, given a set of perturbations of a given image (possibly uncountable), would the classification of voxels change from background to segment and vice versa. Given an set of input images $\mathcal{I} \subset \mathbb{R}^{n_1 \times n_2 \times n_3}$, the set of possible segmentation outputs $\mathcal{O} = \bigcup_{I \in \mathcal{I}} \mathcal{N}(I)$. Given a reference image I_{ref}, a voxel $[i, j, k]$ is said to be fragile if it is classified into different classes in the reference and the perturbations. That is, $\exists I \in \mathcal{I}$ such that $\mathcal{N}(I)[i, j, k] \neq \mathcal{N}(I_{\mathsf{ref}})[i, j, k]$. Given an set $\mathcal{I}$, we define the segmentation fragility as the fraction of voxels that are classified as fragile when compared to the voxels of interest, i.e., the voxels that are classified as segment.

$$fragility = \frac{\#fragile\ voxels}{\#voxels\ of\ interest}.$$

While it might be convenient to count the ratio of fragile voxels to total number of voxels, we propose this definition to better account for signal-to-noise ratio in biomedial image segmentation. Naturally, a fragility of 0 would mean that none of the voxels have fragile segmentation and a fragility of 1 would mean that the noise induced by perturbing the input is equal to that of the size of the segment. Also observe that under increased perturbation, fragility of a set could be more than 1.

We use reachable set computation techniques for computing overapproximation of the image of the classification network on the set of perturbed inputs. Therefore, we might not be able to exactly compute the fragility of a given segmentation network on the set $\mathcal{I}$. We therefore compute underapproximation and overapproximation of the network fragility.

III. Biomedial Image Segmentation Using U-Nets

One of the widely used neural architectures in medical imaging is U-Net. The U-Net architecture was introduced by [22] in 2015 specifically for biomedical image analysis. The architectural diagram as introduced in the seminal paper is shown in Figure 1 . It is somewhat similar in form to a

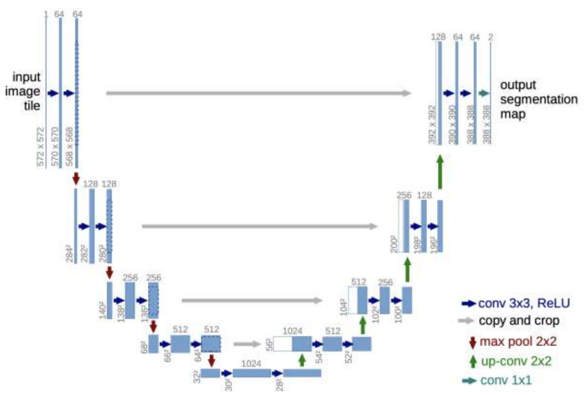


Fig. 1: Original U-Net architecture for biomedical image segmentation as introduced in [22].

convolutional autoencoder, in that the U-Net first performs convolutional downsampling for feature detection like an encoder, then convolutional upsampling to synthesis new information from the detected features like a decoder. However, the U-Net also uses "skip connections" where output from intermediate layer on the left side of the U are saved and provided to layers on the right side of the U in order to provide important higher resolution spatial context for the model to use in its synthesis. It has been shown empirically that U-Nets are significantly better than other architectures comprising of purely convolutional components. While readers might be familiar with most of the components, we briefly explain the components of the U-Net architecture primarily because these deal with 3 dimensional inputs.

Downsampling Convolutions (Conv): A 3 dimensional convolutional kernel is an array of dimensions $l \times w \times d$ denoted as $K[l][w][d]$. Given an input of dimensions $n_1 \times n_2 \times n_3$, $I[n_1][n_2][n_3]$, the convolution of $I{*}K$ is an array of dimensions $(n_1 - l) \times (n_2 - w) \times (n_3 - d)$ and the element defined as

$$(I * K)[i, j, k] = \sum_{\substack{p \in [0, l-1], \\ q \in [0, w-1], \\ r \in [0, d-1]}} I[i + p, j + q, k + r] \cdot K[p, q, r].$$

For generalization, one can add bias that will be added while doing weighted sum during convolution.

Rectified Linear Unit (ReLU): The rectified linear unit acts as a filter that removes all the values less than 0. In an U-Net, a ReLU layer is performed after the convolution operation. Given an input of dimensions $n_1 \times n_2 \times n_3$, the ReLU operation is another tensor of size $n_1 \times n_2 \times n_3$ defined as

$$\mathsf{ReLU}(I)[i, j, k] = \begin{cases} 0 & \textit{if} \quad I[i, j, k] \leq 0. \\ I[i, j, k] & \textit{otherwise}. \end{cases}$$

Skipped Connection: The output tensor of a specific downsampling convolution is saved, concatenated with the output from a later upsampling convolution, and given as input to subsequent convolutional layers.

Upsampling Convolution (ConvTranspose): A 3 dimensional upsampling convolution kernel is an array of size $l \times w \times d$ denoted by $K[l][w][d]$. Given an input I of dimensions $n_1 \times n_2 \times n_3$, $I[n_1][n_2][n_3]$, an upsampling convolution (ConvTranspose) $I{*}{*}K$ is an array of size $(n_1 \cdot l) \times (n_2 \cdot w) \times (n_3 \cdot d)$ as defined as

$$(I * *K)[l \cdot i + p, w \cdot j + q, d \cdot k + r] = I[i, j, k] \cdot K[p, q, r].$$

where $i \in [0, n_1 - 1], j \in [0, n_2 - 1], k \in [0, n_3 - 1]$ and $p \in [0, l - 1], q \in [0, w - 1], r \in [0, d - 1]$. Informally, each voxel of the image I is magnified by a factor of $l \times w \times d$, weighted by the factor defined in the kernel. The ConvTranspose is sometimes presented as convolution of an image after introducing new voxels — whose number is defined by padding and stride — with zero values and then performing the usual convolution operation. It is easy to observe that such variants are mathematically equivalent to a sequence of one more convolution operations that can be described in simple mathematical terms. It is a usual practice that after such upsampling, one would often perform convolutions to smoothen the image.

While in the original U-Net paper, one suggested using MaxPooling layer, empirical results have shown that one could achieve accuracy using U-Nets without MaxPool layer. The last layer in a U-Net is a softmax layer where each voxel is compared to a threshold and is classified as segment if the value is more than threshold and is classified as background if the value is less than the threshold.

IV. Verification of Image-Net Neural Networks Using TensorStars

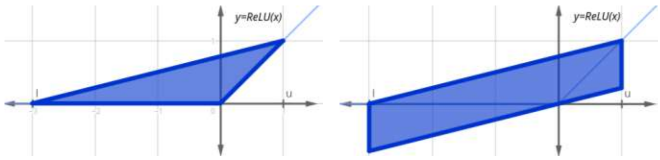
In this section, we present the extension of star sets to 3 dimensional images and call them TensorStars. We introduce the definition of TensorStars, present the operations that overapproximate the Conv , ReLU , Skip , $\mathsf{ConvTrans}$ operations on TensorStars, and present an algorithm to compute fragility of a given perturbation for a reference.

Definition 1: A TensorStar of dimensions $n_1 \times n_2 \times n_3$ is a tuple $\mathcal{T} = \langle a, G, P \rangle$ where a is a 3 dimensional array of dimensions $n_1 \times n_2 \times n_3$, $G = \{g_1, g_2, \ldots, g_m\}$ is a set of m generators where each g_i is a 3 dimensional array of dimensions $n_1 \times n_2 \times n_3$ and P is a predicate $P : \mathbb{R}^m \to \{\top, \bot\}$. The set of tensors defined by a Tensor star is denoted as

$$[\![\mathcal{T}]\!] = \{\, I \mid \exists \alpha_1, \ldots, \alpha_m, I = a + \alpha_1 g_1 + \ldots + \alpha_m g_m \\ \text{and } P(\alpha_1, \ldots, \alpha_m) = \top\}.$$

Informally, a TensorStar is an affine combination of anchor and generator tensor where the coefficient of the anchor is 1 and the coefficients for the generators satisfy the predicate in the TensorStar. We abuse notation and use $\mathcal{T}$ to represent both the tuple representation of a TensorStar and the set represented by the tuple.

In general, there is no restriction on the format of each of the anchors and the generators. However, if one wants to model



(a) Convex Hull (b) Parallelogram

Fig. 2: Overapproximation of a ReLU on a single voxel

voxel level perturbation of an image, each g_i is represented as a 3 dimensional array with all zeros execept for one voxel. Similarly, while the predicate P can be in general any predicate, efficient solvers exist only for only a restricted subset. We often consider the predicate P to be defined as a conjunction of linear constraints.

One of the primary advantages of using the star representation is that performing affine transformations over the set represented by the star can be computed very easily by applying the affine transformation over the anchor and generators. The predicate that captures the complex dependency between the coefficients remains unchanged during affine transformations and needs only to be modified during nonlinear operations such as ReLU .

A. Operations on Tensor Stars

Convolution and Transpose Convolution Operation: Given a convolution kernel K of dimensions $l \times w \times d$, the convolution operation on a TensorStar $\mathcal{T} = \langle a, G, P \rangle$, denoted as $\mathcal{T} * K = \langle a * K, G * K, P \rangle$ where $G * K = \{g_1 * K, \ldots, g_m * K\}$. That is, for computing convolution over TensorStars, we perform convolution operations on anchor and generators.

Lemma 1: Given a TensorStar $\mathcal{T}$ and a convolutional kernel K, $[\![\mathcal{T} * K]\!] = \{I * K \mid I \in [\![\mathcal{T}]\!]\}$. That is, the convolution operation on anchor and generators is equivalent to performing convolution on all the tensors represented by TensorStar.

Proof: Follows easily from the definition of Conv and the affine nature of convolution operation. ∎

Similar to the convolution operation, given an upsampling convolution kernel K of dimensions $l \times w \times d$, the upsampling convolution operation on a TensorStar $\mathcal{T} = \langle a, G, P \rangle$, denoted as $\mathcal{T} * *K = \langle a * *K, G * *K, P \rangle$ where $G * *K = \{g_1 * *K, \ldots, g_m * *K\}$. The correctness of the transformation is very similar to the convolution operation.

ReLU Operation: Several previous works have overapproximated the nonlinear ReLU operation as convex operation. Two of the techniques presented in the literature are the minimal convex hull and parallelotope overapproximation. The minimal convex hull is given in Figures 2a and the parallelogram convex hull is given in Figure 2b.

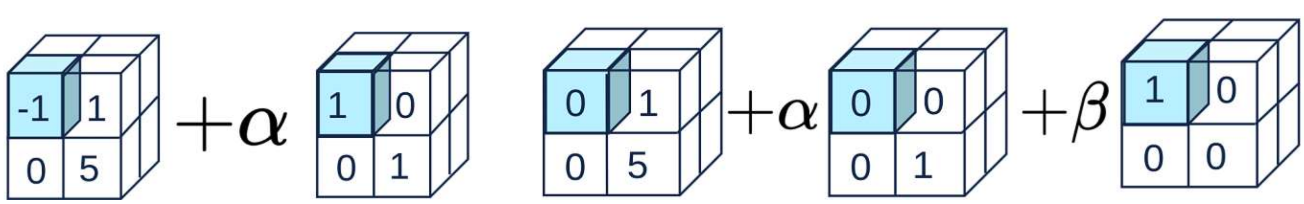
In medical imaging, since ReLU operation is applied for each voxel, we first collect the constraints that determine the possible values assigned for a voxel. That is, for a given voxel $[i, j, k]$ the possible range of values in $\mathcal{T}$ for voxel $[i, j, k]$ is given by the constraints:

$$\mathcal{T}[i, j, k] = \{v | v = a[i, j, k] + \sum_{l=1\ldots m} \alpha_i g[i, j, k] \wedge P(\overline{\alpha}) = \top\}$$

Where $\overline{\alpha}$ is the vector $(\alpha_1, \ldots, \alpha_m)$. Since P is a conjunction of linear constraints, one can compute upper and lower bounds of $\mathcal{T}[i, j, k]$ using linear programming and we represent the upper and lower bounds as $u_{[i,j,k]}$ and $l_{[i,j,k]}$. Once computed, we overapproximate the region as a parallelotope or minimal convex hull. We give the example as a parallelotope by introducing a new variable β and introduce the constraints as follows $\beta \geq 0$; $\beta \geq \mathcal{T}[i, j, k]$; and $\beta \leq \frac{u_{[i,j,k]}\mathcal{T}[i,j,k]}{u_{[i,j,k]} - l_{[i,j,k]}}$. We illustrate this through an example of such transformation over 3 dimensional tensors.

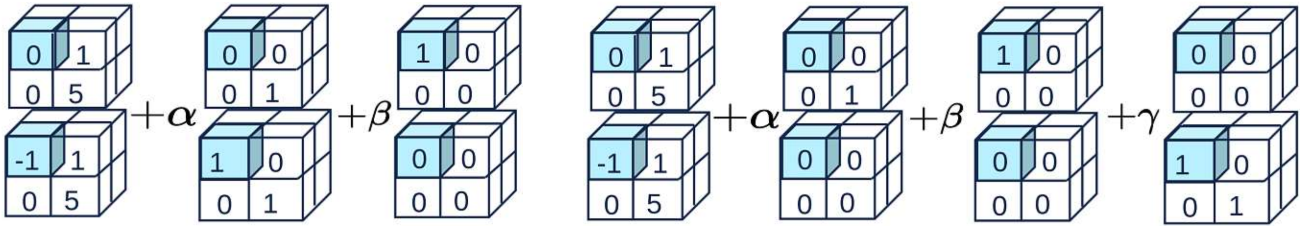
Example 1: Consider the TensorStar as given in Figure 3a where the tensor star consists of an anchor and only one generator and the predicate is defined by constraints $\alpha \in [-2, 2]$. All the values in the anchor and generator that are behind the visible voxels are 0. For the first voxel (highlighted in blue), the upper and lower bound is given as $\max\{-1 + \alpha \,|\, \alpha \in [-2, 2]\}$ and $\min\{-1 + \alpha | \alpha \in [-2, 2]\}$. Therefore $u = 1; l = -3$. Since the upper and lower bounds have intersection with values both below and above 0, we change the corresponding index in the anchor tensor and make it 0. Further, since after the ReLU transformation we model the values in the particular voxel using the variable β, we make the corresponding voxel value of the first generator to be 0. We then add the additional constraints on β as $\beta \geq 0$; $\beta \geq -1 + \alpha$; and $\beta \leq \frac{1((-1+\alpha)-(-3))}{1-(-3)}$. The last constraint can be rewritten as $\beta \leq \frac{\alpha+2}{4}$.

In this specific example, only one of the voxel values has a nonempty intersection with values greater than and less than 0. Therefore, we only introduce one new variable. In theory, all the voxels in the TensorStar can potentially have nonempty intersection with values greater than and less than 0. In such cases, we have to introduce $n_1 \times n_2 \times n_3$ new variables and add thrice the number of constraints — that can clearly cause an explosion in the number of constraints and variables. However, in practice, only a few voxels are capable of achieving both



(a) TensorStar before ReLU . (b) TensorStar after ReLU .

Fig. 3: Overapproximation of a ReLU on a TensorStar.



(a) Concatenation of TensorStars with variable preserving

(b) Concatenation of TensorStars with variable renaming

Fig. 4: Concatenation Operation from Skip for TensorStars.

negative and positive values and therefore we do not suffer from the explosion of such variables.

Lemma 2: Minimal Convex Hull and Parallelogram overapproximations of the TensorStar compute the overapproximation of ReLU operation on the set of tensors represented by the TensorStar.

Concatenation From Skipped Connection Operation: In a U-Net, the values of voxels after a convolution layer are concatenated together with the voxels obtained after upsampling. In theory, one can concatenate two 3 dimensional images of any sizes where one would add sufficient padding around the smaller image and make sure that both the images have compatible dimensions. In practice, we rarely develop such U-Nets and instead make sure that the dimensions of the images being concatenated is exactly the same.

Observe that the TensorStar obtained after the upsampling has been transformed through several ReLU layers as opposed to the TensorStar from the skipped connection. Therefore, the TensorStar from the upsampling can potentially have more variables — and therefore, more generators. This begs the question, how to concatenate two tensor stars that do not have the same number of generators.

There are two possible ways of concatenating such TensorStars. In the first technique, we preserve the mapping from the generator to the corresponding coefficient and concatenate the generators for the same coefficient. The second technique is to forget the link between the generators and the corresponding coefficients and instead generate new set of variables. Since a ReLU operation only creates and adds new variables, the original constraints on the older variables remain.

Example 2: Consider the two TensorStars represented in Figure 3a and Figure 3b. The two ways to concatenate the TensorStars are given in Figure 4. In Figure 7a, we preserve the relationship between the coefficient variable α and concatenate the two corresponding generators. Since the first tensor star does not have a corresponding generator for the coefficient β, the corresponding generator is concatenated with 0s. In Figure 7c, we do not preserve the relationship between the generators. Instead, we create a new variable γ and copy the constraints into a new variable.

Consider two TensorStar $T_1 = \langle a_1, G_1, P_1 \rangle$ and $T_2 = \langle a_2, G_2, P_2 \rangle$ where $\mathcal{T}_1$ is from the skipped connection with $G_1 = \{g_1, g_2, \ldots, g_{k_1}\}$ and $G_2 = \{g'_1, g'_2, \ldots, g'_{k_2}\}$ where G_1 and G_2 are determined by the order of the coefficient variables with $k_2 \geq k_1$ and all the constraints in P_1 are also part of P_2.

The variable preserving way to concatenate $\mathcal{T}_1$ and $\mathcal{T}_2$ is given as $\mathcal{T}_1 \odot_p \mathcal{T}_2 = \langle a_1 \odot a_2, G_1 \odot_c G_2, P_2 \rangle$ where $G_1 \odot_c G_2 = \{g_1 \odot g'_1, g_2 \odot g'_2, \ldots, g_{k_1} \odot g'_{k_1}, \mathbf{0} \odot g'_{k_1+1}, \ldots, \mathbf{0} \odot g'_{k_2}\}$. Here $\odot$ operation is overloaded to be concatenation operation on 3 dimensional arrays. That is, we concatenate the generators corresponding to the same coefficient variables and for all the coefficient variables that do not have the corresponding generators in $\mathcal{T}_1$, we concatenate $\mathbf{0}$.

The variable renaming way to concatenate $\mathcal{T}_1$ and $\mathcal{T}_2$ is given as $\mathcal{T}_1 \odot_r \mathcal{T}_2 = \langle a_1 \odot a_2, G_1 \odot_r G_2, P'_1 \odot_r P_2 \rangle$ where $G_1 \odot_r G_2 = \{g_1 \odot \mathbf{0}, \ldots, g_{k_1} \odot \mathbf{0}, \mathbf{0} \odot g'_1, \ldots, \mathbf{0} \odot g'_{k_2}\}$ and $P_1 \odot_r P_2$ renames all the variables in P_1 to fresh variables. Notice that variable renaming way would increase the number of generators to be $k_1 + k_2$ as opposed to k_2 in the variable preserving way. While it would seem obvious that one would want to reduce the number of variables, in some instances, the number of generators with all 0 values becomes large. Therefore, for improving the efficiency of 3 dimensional array multiplication, we drop the generators with all zero coefficients and use variable renaming.

Lemma 3: Given TensorStars $\mathcal{T}_1$ and $\mathcal{T}_2$, the variable preserving and variable renaming concatenation denoted as $\mathcal{T}_1 \odot_p \mathcal{T}_2$ and $\mathcal{T}_1 \odot_r \mathcal{T}_2$ compute the overapproximation of the concatenation from Skip in the U-Net.

Robustness Verification of U-Nets using TensorStars: The algorithm for computing an overapproximation of the input TensorStar when operated on by a neural network is provided in Algorithm 1. The algorithm applies the corresponding transformation on the TensorStar as presented earlier in this section. After computing the overapproximation of the a TensorStar with the five different componenents, we finally get the TensorStar representation before the softmax layer. For each voxel, we compute the upper and lower bound of the threshold values for the voxel and compare them with the threshold. If both the upper and lower bound are above the threshold, we assign that the voxel is classified as the segment; similar for both the values below the threshold. If the upper bound of the voxel is above the threshold and the lower bound is below the threshold, we label the corresponding voxel as unstable.

Algorithm 1: Algorithm that computes an overapproximation of the output when the given input TensorStar is operated on by the input Neural Network.

input : Neural Network: $\mathcal{N}$ obtained by sequential composition of components $\mathcal{N}_1, \ldots, \mathcal{N}_k$; Input TensorStar $\mathcal{I}$.
output : A TesorStar $\mathcal{O}$ that is an overapproximation of $\mathcal{N}(\mathcal{I})$.

$\mathcal{O}_0 \leftarrow I$;
for *each i from 1 to k* **do**
  if $\mathcal{N}_i$ *is* ***Conv*** **then**
    $\mathcal{O}_i \leftarrow \mathcal{O}_{i-1} * \mathcal{N}_i$;
    `// Convolution on TensorStar`
  if $\mathcal{N}_i$ *is* ***ConvTranspose*** **then**
    $\mathcal{O}_i \leftarrow \mathcal{O}_{i-1} * * \mathcal{N}_i$;
    `// Upsamping Convolution`
  if $\mathcal{N}_i$ *is* ***ReLU*** **then**
    $\mathcal{O}_i \leftarrow \mathsf{ReLU}(\mathcal{O}_{i-1})$;
    `// ReLU on TensorStar`
  if N_i *is* ***SkipConnection*** **then**
    Create a copy of $\mathcal{O}_{i-1}$;
    $\mathcal{O}_i \leftarrow \mathcal{O}_{i-1}$;
    `// Skip connections stores a copy that will be used during concatenation.`
  if N_i *is* ***Concatenation*** *of $\mathcal{N}_{i-1}$ and $\mathcal{N}_j$* **then**
    $\mathcal{O}_i \leftarrow \mathcal{O}_{i-1} \odot \mathcal{O}_j$;
    `// Concatenation with or without variable renaming as determined by the user performing verification.`
return $\mathcal{O}_k$;

V. Leveraging Accelerators and Parallelism for Improved Efficiency of Tensor Star Verification

There are two main inefficiencies in the overapproximation of TensorStars for the neural network layers. First, the typical size of modern medical image is greater $128 \times 128 \times 128$. In this study, we downsample images to $32 \times 32 \times 32$. Therefore, each convolution operation would have to perform 32^3 3 dimensional array multplication operations for each anchor and generator in the TensorStar. Similarly, the ReLU operation is performed on each and every voxel after every convolutional layer, therefore, the ReLU overapproximation would have to perform approximately $32^3 \approx 40,000$ linear programs for computing the upper and lower bounds. Given the large number of these operations, any fractional improvement in the performance of these two operations, would have a large impact on the overall performance of the robustness verification. *We have implemented two performance enhancements and as a result, have achieved 10x improvement in the time taken for verification.*

GPU Acceleration for Tensor Multiplications: One of the primary reasons for wide-scale usage of Graphical Processing Units for machine learning is their efficiency at performing tensor products. All purely affine transformations were accelerated on the GPU using TensorFlow's Keras backend [23].

Parallelization of ReLU Overapproximation: At each voxel v, determining the overraproximate reachable region of the ReLU involves solving two linear programs determined by the TensorStar predicate to find u, and l, the upper and lower bounds of the set reachable at v. These linear programs can be solved independently for each voxel, hence can profit from program parallelism. This is called leveraging the *embarassing parallelism* in the algorithm.

We accelerated the ReLU computations by integrating python's multiprocessing library [24]. Naive implementation in the multiprocessing library would cause a lot of busy-waiting on concurrent data structures and do not improve the performance. To overcome this, we save all the constraints on coefficient variables, anchors, and generators in a shared memory. Then each individual thread will read the corresponding coefficients for each ReLU and formulate a local linear program that invokes the python linear programming library. The results from each individual linear program is written to a thread specific write locations, avoiding the usage of global lock on concurrent data structure. This is represented diagrammatically in Figure 5. This acceleration achieves a greater than $10\times$ speed up in sample verification instances, as shown in table I.

I	**A**	**Param**	**AT (s)**	**RT (s)**
35	D	(240, 0.005)	2630	20113
35	D	(240, 0.015)	2923	21372
35	D	(180, 0.005)	3099	39324
35	D	(180, 0.010)	7062	136539
35	P	(18, 10, 12) - 1 - [100,102]	2723	22598
36	D	(240, 0.005)	2639	21215
36	D	(240, 0.015)	2704	22547
36	P	(18, 10, 12) - 1 - [100,102]	2731	24153
37	D	(240, 0.005)	2659	22018
37	D	(240, 0.015)	3764	32637
37	P	(18, 10, 12) - 1 - [100,102]	2723	22270
Average Speedup			**10.79**	

TABLE I: Speed up using accelerators for neural network verification. Legend: **I**: Reference Image ID; **A**: Perturbation type D - Darkening, B - Beam Hardening, P - Patch; textbfParam: parameters for perturbation — Threshold and ϵ for darkening, min and max intensity and the layers for beam hardening, location, patch size of $n \times n \times n$ and intensity range for patch; **RT**: running time without acceleration in seconds; **AT**: running time with acceleration in seconds

VI. Evaluation

To demonstrate the effectiveness of the robustness verification using TensorStars, we verify the robustness of a U-Net performing Kidney segmentation.

A. Training of Kidney Segmentation Network

As described in Section I, in the domain of medical image segmentation, there is a lot more emphasis on the architecture

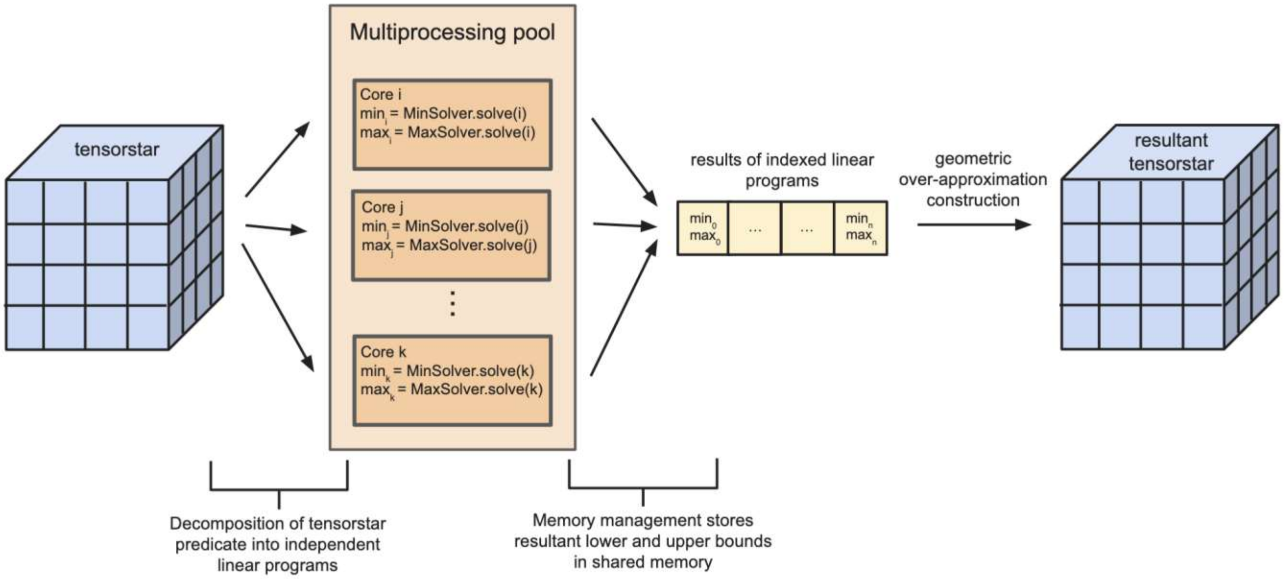


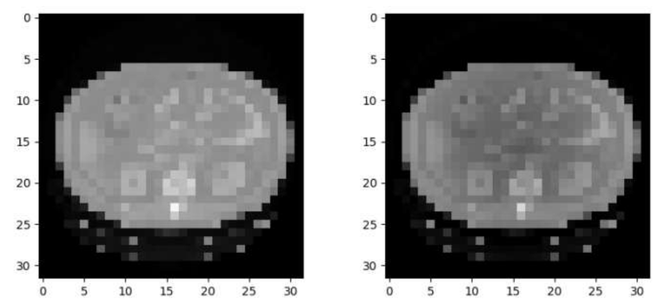
Fig. 5: Multiprocessor acceleration of ReLU overapproximation computation. MinSolver and MaxSolver are cvxpy based linear program instances giving the minimum and maximum, respectively, achievable value of a given voxel.

and training technique as opposed to the specific network used in segmentation. Therefore, we trained our own Kidney segmentation network with the 3D UNet as described in [25] implemented in Python 3.7 with TensorFlow2, with minor modifications; specifically, restricting to ReLU activation functions and removing pooling for simplicity. The model was trained on 30 randomly selected cases from the KITS datasets [26] with the standard data supplementation techniques. While the standard image sizes are $128 \times 128 \times 128$, we downsampled image sizes to $32 \times 32 \times 32$ to make the training and verification more efficient. The total number of resulting ReLU operations in this network are $\approx 460,000$. We used categorical cross entropy for loss function and used the ADAM optimizer trained for 1000 epochs on a machine with Intel Core i7-11700F Rocket Lake 2.5GHz (4.9GHz Turbo) 8-Core 16-Thread CPU and NVIDIA GeForce RTX 3060 Ti 8GB GPU. The final value of loss function was less than 0.001.

B. Perturbation of Input Using Noise Models

We consider three main types of perturbations on a given input: *darkening*, *beam hardening*, and *patch*.

Darkening Perturbation: For this perturbation, we select an intensity threshold T and for all the voxels of intensity greater than T, the intensity is reduced by a factor of $[0, \epsilon]$ where $\epsilon \in \{0.005, 0.01, 0.015\}$. That is, if a given voxel has intensity of 200 with a threshold of $T = 180$ and $\epsilon = 0.005$, then the intensity of the voxel in the TensorStar is restricted to be in the interval $[0, 0.005 \times 200] = [0, 1]$. Given a reference image, this perturbation can be modeled as a tensor star where the anchor is the reference image except for the voxels with intensity above T made 0. For each voxel erased from the anchor, we add a corresponding generator where only that voxel is given a value 1 and all other voxels are 0. The coefficient of the corresponding generator is the range of allowed intensities defined by the darkening factor ϵ.



(a) Reference (b) Beam Hardening

Fig. 6: An example of Beam Hardening with decrease in intensity at the center by 100 units.

Beam Hardening Perturbation: Biomedical researchers have observed several distortions in the intensity of x-rays when they pass through through human tissue as early as 1970s [27]. One such widely documented noise is Beam Hardening and more specifically cupping effect where low intensity beams fail to reflect from the center of the image. As a result, the values of intensity appears to be lower at the center of the image but remain unchanged at the edges of the image. While computational techniques to account for beam hardening have been known in the literature [28], [29], we intend to check whether a given neural network is robust with respect to beam hardening perturbation of a given intensity. To model such cupping effect, we introduce a maximum and minimum decrease in the intensity of the center of the image $[min_c, max_c]$ and the gradient is uniform such that the edges of the image have no loss in intensity. Given a reference image, this perturbation can be modeled with the reference image

I	A	# K	# U	# M	F	# IG	# FG	Param	RT
35	D	168	0	1	0.006	1	18	(240,0.005)	0::44
35	D	168	0	1	0.006	1	57	(240,0.01)	0::44
35	D	168	0	1	0.006	1	85	(240,0.015)	0::49
35	D	166	0	5	0.03	4	149	(200,0.005)	0::57
35	D	166	0	5	0.03	4	285	(200,0.01)	2::45
35	D	166	0	5	0.03	4	450	(200,0.015)	7::58
35	D	165	1	5	0.03	6	156	(180,0.005)	0::52
35	D	165	1	5	0.03	6	344	(180,0.01)	1::58
35	D	165	1	5	0.03	6	502	(180,0.015)	4::22
35	D	158	1	41	0.25	50	1016	(128,0.005)	35::20
35	D	-	-	-	-	-	-	(128,0.01)	TO
35	B	158	1	18	0.11	1	821	(50,51) - [12,13]	18::01
35	B	-	-	-	-	-	-	(50,150) - [8,12]	OOM
35	P	169	0	2	0.006	1	53	(18,10,12) - 1 - [100,102]	0::45
35	P	173	3	26	0.17	3	613	(18,10,12) - 3 - [100,102]	17::38
35	P	-	-	-	-	5	-	(18,10,12) - 5 - [100,102]	TO
36	D	151	0	0	0.0	1	33	(240,0.005)	0::44
36	D	151	0	0	0.0	1	48	(240,0.01)	0::44
36	D	151	0	0	0.0	1	57	(240,0.015)	0::45
36	D	148	0	5	0.03	6	208	(200,0.005)	1::17
36	D	148	0	5	0.03	6	337	(200,0.01)	3::52
36	D	148	0	5	0.03	6	518	(200,0.015)	14::47
36	B	125	1	27	0.19	1	1278	(50,51) - [12,13]	83::31
36	B	-	-	-	-	-	-	(50,150) - [8,12]	OOM
36	P	152	0	1	0.006	1	58	(18,10,12) - 1 - [100,102]	0::46
36	P	126	3	39	0.28	3	1118	(18,10,12) - 3 - [100,102]	159::07
36	P	-	-	-	-	5	-	(18,10,12) - 5 - [100,102]	TO
37	D	77	0	1	0.012	2	43	(240,0.005)	0::44
37	D	77	0	1	0.012	2	74	(240,0.01)	0::49
37	D	77	0	1	0.012	2	107	(240,0.015)	1::03
37	D	78	0	2	0.024	5	113	(200,0.005)	1::00
37	D	78	1	2	0.024	5	194	(200,0.01)	2::02
37	D	78	0	2	0.024	5	296	(200,0.015)	5::52
37	B	75	0	25	0.33	1	837	(50,51) - [12,13]	15::12
37	B	-	-	-	-	-	-	(50,150) - [8,12]	OOM
37	P	78	0	2	0.024	1	44	(18,10,12) - 1 - [100,102]	0::46
37	P	74	1	14	0.20	3	787	(18,10,12) - 3 - [100,102]	41::40
37	P	-	-	-	-	5	-	(18,10,12) - 5 - [100,102]	TO

TABLE II: Legend: **I**: Reference Image ID; **A**: Perturbation type D - Darkening, B - Beam Hardening, P - Patch; **# K**: # of voxels classified as Kidney in reference image; **# U**: # of unstable voxels; **# M**: # of provably misclassified voxels; **F:** Fragility of the network on perturbation; **# IG**: # of generators in the initial TensorStar; **# FG**: # of generators in the final TensorStar; **Param**: parameters for perturbation — Threshold and ϵ for darkening, min and max intensity and the layers for beam hardening, location, patch size of $n \times n \times n$ and intensity range for patch; **RT**: running time in hrs::min;

as the anchor, and one generator where the coefficients of generator models the uncertainty in the decrease of intensity in the interval $[min_c, max_c]$ and the corresponding generator is a uniform gradient with coefficient 1 in the center that gradually tapers to 0 at the edges of the image. An example of one layer of an image and its corresponding beam hardening perturbation is given in Figure 6.

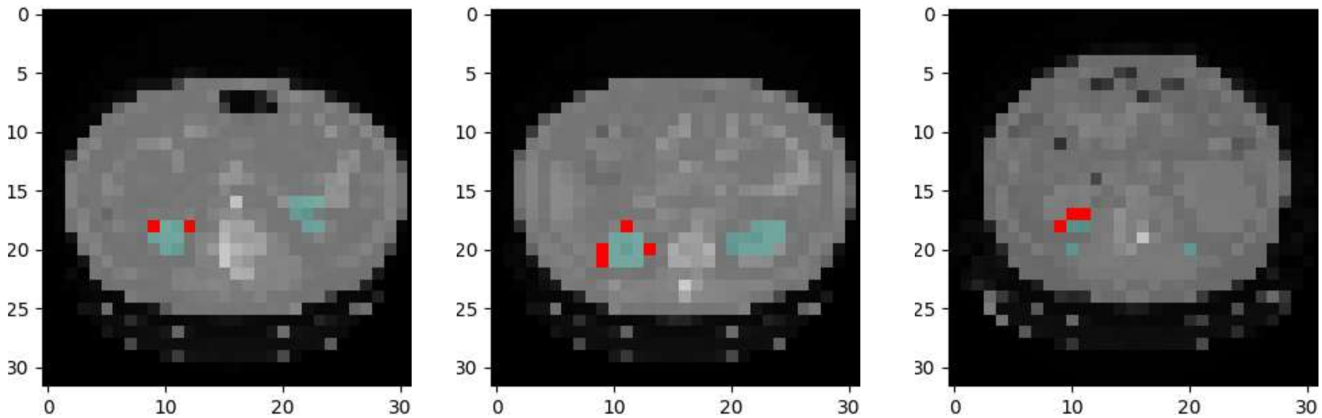
Patch Perturbation: In this perturbation, we select a patch of voxels of a specific size, say 3×3, and modify the intensity in these voxels to be determined by a specific range. Similar to darkening perturbation, patch perturbation can also be modeled using TensorStars.

C. Verification Results

After computing the overapproximation of TensorStar with U-Net, we compute the number of guaranteed correctly classified voxels, number of unstable voxels, and number of guaranteed misclassified voxels. All of our experiments have been performed on an Ubuntu 22.04 system with 32 Core CPU (AMD Ryzen Threadripper PRO 7975WX Processor (5.3 GHz, 32 Cores)) and 256 GB main memory with NVIDIA Geforce RTX 4090 with 24GB VRAM Memory. The results of the experiments on different types of perturbations and the time taken for such verification is provided in Table II. Examples of images with unstable, misclassified, and correctly classified voxels is given in Figure 7 (in Appendix). Since the actual images are 3 dimensional, the visual representation of all the guaranteed misclassified and unstable voxels is available here [1].

We present some observations from the verification results. First is that the amount of time for verification would definitely depend on the specific reference image and the type of perturbation. Second, the darkening perturbation is the easiest to verify whereas the beam hardening and patch are often computationally intensive. In some instances, the fragility of the neural network is around 0.2 and in our experiments going beyond this value requires significant computational resources. Third, while some of the misclassification of voxels is happening at the places where perturbation is introduced,

[1]https://tinyurl.com/h8mpj9t8



(a) Darkening (b) Beam Hardening (c) Patch

Fig. 7: Medical images with provably incorrectly classified voxels labelled in red and the Kidney labelled in green.

in several instances, we observed misclassification in layers that are directly not perturbed. This is specifically in the case of Beam Hardening where introducing uncertainty in only one layer introduced misclassification in multiple layers of the image. Finally, while there are instances where we could not finish verification in a week (TO: for time out in the results table), we encountred a new bottleneck for verification. In various beam hardening perturbations we realized that the system ran out of memory (OOM: system ran out of memory and crashed in results table). This could be possibly because of two reasons. First, as the size of the perturbations and constraints grew, the memory required for storing the TensorStar in shared memory also grew. Second, the memory requirements for solving the linear programs also grew with the size of the linear program.

D. Comparison with Other Tools and Verification Benchmarks

Techniques for verification of neural networks can be broadly classified into one of two classes: constraint satisfaction driven or representation driven. Examples of constraint satisfaction driven works are Reluplex [20], β-Crown [30], $\alpha - \beta$ Crown [31], Marabou [32], Sherlock [33], Planet [34], etc. In these works, the goal is to formulate the robustness verification as a constraint solving problem and develop general decision procedures for accelerating the search for robustness violation or satisfaction. In the representation driven techniques for robustness verification, the goal is to compute an explicit representation of possible outcomes of a given input set. The types of representations used vary from Zonotopes, to Polyhedra, to Star sets. Examples of tools that use this technique are ERAN [35], [36], DeepPoly [37], NNV [38], and verification using ImageStars [21], and others [39], [40]. For a full review of current state-of-the-art methods of deep learning model verification see [41]. All of the verification tools currently use VNN-LIB [42] for specifying properties.

While in theory the tools for verifying robustness properties of neural networks can work with input of any dimensions, in practice these tools support limited verification instances. First, the representation based techniques do not support 3 dimensional input data and skipped connections features that are prominent in U-Nets. Despite our best efforts, we could not use $\alpha - \beta$ Crown for verifying robustness of 3 dimensional U-Net and have received confirmation from the developers that they have not tested the $\alpha - \beta$ Crown on 3 dimensional inputs. Additionally, beam-forming, a widely documented source of noise in medical imaging cannot be represented using the current VNN-LIB standard. This is because beam forming requires correlating the perturbations of different voxels using a common variable. Hence, our work fills an important and necessary gap in the domain of robustness verification of neural networks.

VII. Conclusion and Future Work

While the current work falls in the category of robustness verification using representation, it differs from the literature in three key aspects. First, none of these works have been specifically tailored to U-Nets. The specific three dimensional nature of the input and output modalities require special purpose implementation that is currently not achieved by other representation based tools. Second, due to the high dimensionality of medical imaging data, the number of ReLU operations encountered in these verifications is at least an order of magnitude larger than neural networks for 2D images trained on CIFAR dataset. Without leveraging the parallelism and hardware acceleration, verifying robustness of medical imaging networks would be unachieveable. Finally, unlike purely synthetic perturbations of input images, we demonstrated that verification techniques can be used to verify real noise profiles such as Beam Hardening.

To the best of authors knowledge, this is the first paper on verifying robustness of neural networks on medical imaging data. We have shown that extension of star based techniques, together with enhancements can indeed verify robustness of realistic perturbations that can be observed in practice. While we have demonstrated the robustness (or the lack thereof) for U-Nets, the size of the U-Nets considered in this paper are not fully applied in clinical setting. Our goal is to scale the verification to high resolution medical images of $512\times512\times512$ which would require improving the efficiency by at least three orders of magnitude.

Acknowledgements: The authors would like to acknowledge the support from AFOSR under award number FA9550-23-1-0286 and NSF under awards 2038960 and 2303564. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFOSR or NSF.

REFERENCES

[1] Nima Tajbakhsh, Laura Jeyaseelan, Qian Li, Jeffrey N. Chiang, Zhihao Wu, and Xiaowei Ding. Embracing imperfect datasets: A review of deep learning solutions for medical image segmentation. *Medical Image Analysis*, 63:101693, July 2020.

[2] Fang Lu, Fa Wu, Peijun Hu, Zhiyi Peng, and Dexing Kong. Automatic 3D liver location and segmentation via convolutional neural network and graph cut. *International Journal of Computer Assisted Radiology and Surgery*, 12(2):171–182, February 2017.

[3] Fang Liu, Zhaoye Zhou, Hyungseok Jang, Alexey Samsonov, Gengyan Zhao, and Richard Kijowski. Deep convolutional neural network and 3D deformable approach for tissue segmentation in musculoskeletal magnetic resonance imaging. *Magnetic Resonance in Medicine*, 79(4):2379–2391, April 2018.

[4] Yabo Fu, Yang Lei, Tonghe Wang, Kristin Higgins, Jeffrey D. Bradley, Walter J. Curran, Tian Liu, and Xiaofeng Yang. LungRegNet: An unsupervised deformable image registration method for 4D-CT lung. *Medical Physics*, 47(4):1763–1774, April 2020.

[5] Xiao Yang, Roland Kwitt, Martin Styner, and Marc Niethammer. Quicksilver: Fast predictive image registration – A deep learning approach. *NeuroImage*, 158:378–396, September 2017.

[6] Boah Kim, Dong Hwan Kim, Seong Ho Park, Jieun Kim, June-Goo Lee, and Jong Chul Ye. CycleMorph: Cycle consistent unsupervised deformable image registration. *Medical Image Analysis*, 71:102036, July 2021.

[7] Tongxue Zhou, Su Ruan, and Stéphane Canu. A review: Deep learning for medical image segmentation using multi-modality fusion. *Array*, 3-4:100004, September 2019.

[8] Yi Li, Junli Zhao, Zhihan Lv, and Jinhua Li. Medical image fusion method by deep learning. *International Journal of Cognitive Computing in Engineering*, 2:21–29, June 2021.

[9] Lequan Yu, Zhicheng Zhang, Xiaomeng Li, and Lei Xing. Deep Sinogram Completion With Image Prior for Metal Artifact Reduction in CT Images. *IEEE Transactions on Medical Imaging*, 40(1):228–238, 2021.

[10] Haofu Liao, Wei-An Lin, S. Kevin Zhou, and Jiebo Luo. ADN: Artifact Disentanglement Network for Unsupervised Metal Artifact Reduction. *IEEE Transactions on Medical Imaging*, 39(3):634–643, March 2020.

[11] P. Herent, B. Schmauch, P. Jehanno, O. Dehaene, C. Saillard, C. Balleyguier, J. Arfi-Rouche, and S. Jégou. Detection and characterization of MRI breast lesions using deep learning. *Diagnostic and Interventional Imaging*, 100(4):219–225, April 2019.

[12] Ke Yan, Youbao Tang, Yifan Peng, Veit Sandfort, Mohammadhadi Bagheri, Zhiyong Lu, and Ronald M. Summers. MULAN: Multitask Universal Lesion Analysis Network for Joint Lesion Detection, Tagging, and Segmentation. In Dinggang Shen, Tianming Liu, Terry M. Peters, Lawrence H. Staib, Caroline Essert, Sean Zhou, Pew-Thian Yap, and Ali Khan, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, pages 194–202, Cham, 2019. Springer International Publishing.

[13] W. Zhao, D.P. Capaldi, C. Chuang, and L. Xing. Fiducial-Free Image-Guided Spinal Stereotactic Radiosurgery Enabled Via Deep Learning. *International Journal of Radiation Oncology*Biology*Physics*, 108(3):e357, November 2020.

[14] Jean-Francois Rajotte, Robert Bergen, David L. Buckeridge, Khaled El Emam, Raymond Ng, and Elissa Strome. Synthetic data as an enabler for machine learning applications in medicine. *iScience*, 25(11):105331, November 2022.

[15] Jelmer M. Wolterink, Anna M. Dinkla, Mark H. F. Savenije, Peter R. Seevinck, Cornelis A. T. Van Den Berg, and Ivana Išgum. Deep MR to CT Synthesis Using Unpaired Data. In Sotirios A. Tsaftaris, Ali Gooya, Alejandro F. Frangi, and Jerry L. Prince, editors, *Simulation and Synthesis in Medical Imaging*, volume 10557, pages 14–23. Springer International Publishing, Cham, 2017. Series Title: Lecture Notes in Computer Science.

[16] Ahmed Alaa, Boris Van Breugel, Evgeny S. Saveliev, and Mihaela van der Schaar. How Faithful is your Synthetic Data? Sample-level Metrics for Evaluating and Auditing Generative Models. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 290–306. PMLR, July 2022.

[17] Liesbeth Vandewinckele, Michaël Claessens, Anna Dinkla, Charlotte Brouwer, Wouter Crijns, Dirk Verellen, and Wouter Van Elmpt. Overview of artificial intelligence-based applications in radiotherapy: Recommendations for implementation and quality assurance. *Radiotherapy and Oncology*, 153:55–66, December 2020.

[18] Avital Oliver, Augustus Odena, Colin A Raffel, Ekin Dogus Cubuk, and Ian Goodfellow. Realistic Evaluation of Deep Semi-Supervised Learning Algorithms. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[19] John R. Zech, Marcus A. Badgeley, Manway Liu, Anthony B. Costa, Joseph J. Titano, and Eric Karl Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study. *PLOS Medicine*, 15(11):e1002683, November 2018.

[20] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, volume 10426, pages 97–117. Springer International Publishing, Cham, 2017.

[21] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. Verification of Deep Convolutional Neural Networks Using ImageStars. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, volume 12224, pages 18–42. Springer International Publishing, Cham, 2020.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, volume 9351, pages 234–241. Springer International Publishing, Cham, 2015.

[23] François Chollet et al. Keras. https://keras.io, 2015.

[24] Python. multiprocessing — process-based parallelism — python 3.8.3rc1 documentation.

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[26] Nicholas Heller, Fabian Isensee, Klaus H Maier-Hein, Xiaoshuai Hou, Chunmei Xie, Fengyi Li, Yang Nan, Guangrui Mu, Zhiyong Lin, Miofei Han, et al. The state of the art in kidney and kidney tumor segmentation in contrast-enhanced ct imaging: Results of the kits19 challenge. *Medical image analysis*, 67:101821, 2021.

[27] Rodney A Brooks and Giovanni Di Chiro. Beam hardening in x-ray reconstructive tomography. *Physics in medicine & biology*, 21(3):390, 1976.

[28] Gabor T Herman. Correction for beam hardening in computed tomography. *Physics in Medicine & Biology*, 24(1):81, 1979.

[29] Elke Van de Casteele, Dirk Van Dyck, Jan Sijbers, and Erik Raman. An energy-based beam hardening model in tomography. *Physics in Medicine & Biology*, 47(23):4181, 2002.

[30] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021.

[31] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-

propagation-based neural network verification. *Advances in Neural Information Processing Systems*, 2022.

[32] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, volume 11561, pages 443–452. Springer International Publishing, Cham, 2019. Series Title: Lecture Notes in Computer Science.

[33] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output Range Analysis for Deep Feedforward Neural Networks. In Aaron Dutle, César Muñoz, and Anthony Narkawicz, editors, *NASA Formal Methods*, volume 10811, pages 121–138. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[34] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, pages 269–286. Springer, 2017.

[35] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting Robustness Certification of Neural Networks. In *International Conference on Learning Representations (ICLR)*. 2019.

[36] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. PRIMA: Precise and General Neural Network Certification via Multi-Neuron Convex Relaxations, 2021. _eprint: 2103.03638,.

[37] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, January 2019.

[38] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, volume 12224, pages 3–17. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.

[39] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, San Francisco, CA, May 2018. IEEE.

[40] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the Single Neuron Convex Barrier for Neural Network Certification. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[41] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J. Kochenderfer. Algorithms for Verifying Deep Neural Networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021.

[42] Stefano Demarchi, Dario Guidotti, Luca Pulina, Armando Tacchella, Nina Narodytska, Guy Amir, Guy Katz, and Omri Isac. Supporting standardization of neural networks verification with vnnlib and coconet. In *FoMLAS@ CAV*, pages 47–58, 2023.