IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 28, 2020, accepted October 30, 2020, date of publication November 23, 2020,
date of current version December 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3039833

Challenges for the Repeatability of Deep Learning Models

SAEED S. ALAHMARI[1,2], (Graduate Student Member, IEEE), DMITRY B. GOLDGOF[1], (Fellow, IEEE), PETER R. MOUTON[1,3], AND LAWRENCE O. HALL[1], (Fellow, IEEE)

[1]Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620-9951, USA
[2]Department of Computer Science, Najran University, Najran 66462-4207, Saudi Arabia
[3]SRC Biosciences, Tampa, FL 33606, USA

Corresponding author: Saeed S. Alahmari (saeed3@usf.edu)

This work was supported by the National Science Foundation under Award NSF Phase I Grant 1746511, NSF Phase II Grant 1926990, and NSF Grant 1513126.

ABSTRACT Deep learning training typically starts with a random sampling initialization approach to set the weights of trainable layers. Therefore, different and/or uncontrolled weight initialization prevents learning the same model multiple times. Consequently, such models yield different results during testing. However, even with the exact same initialization for the weights, a lack of repeatability, replicability, and reproducibility may still be observed during deep learning for many reasons such as software versions, implementation variations, and hardware differences. In this article, we study repeatability when training deep learning models for segmentation and classification tasks using U-Net and LeNet-5 architectures in two development environments Pytorch and Keras (with TensorFlow backend). We show that even with the available control of randomization in Keras and TensorFlow, there are uncontrolled randomizations. We also show repeatable results for the same deep learning architectures using the Pytorch deep learning library. Finally, we discuss variations in the implementation of the weight initialization algorithm across deep learning libraries as a source of uncontrolled error in deep learning results.

INDEX TERMS Deep learning, Pytorch, torch, Keras, TensorFlow, reproducibility, reproducible, repeatability, replicability, replicable deep learning models, deterministic models, determinism.

I. INTRODUCTION

Deep learning has shown great success in many different fields and applications such as image recognition [1], object and keypoints detection [2], [3], pose estimation [4], [5], speech recognition [6] and object segmentation [7]. Many factors have contributed to the success of deep learning, which include data availability, high speed computation resources, and the development of deep learning libraries. Training a deep neural network starts with initialization of the weights in the neural network. Purely random initialization leads to networks that cannot be successfully trained in many cases. Hence, algorithms, which still have a random component, are used to initialize the weights. Then, during training of a deep model, the weights are adjusted through forward and back-propagation steps. This process continues for multiple iterations to minimize a cost function.

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

Since the initial network weights are important for model convergence and eventually model performance, many algorithms have been proposed to optimize initial weights such as the Glorot algorithm [8]. However, to enable repeatable and replicable training results, a manual seed value is needed to reach the same results and eliminate variations during training of deep learning models. Although, initializing network weights while setting a manual seed value produces exact initial weights, there are other factors that prevent learning the same final model each time training is done. These factors prevent achieving the same results for the same test set even when using the same hyper-parameters and hardware and software configurations for a particular software implementation, thus creating a repeatability challenge for deep learning.

Repeatable, replicable, and reproducible methods and findings are very important for scientific research [9], [10] [11]. Other researchers have discussed approaches to reproduce machine learning computational experiments. In [12] there

was a discussion of how to facilitate reproducing the same experiment using computational metadata. Recording and exploiting provenance information is important for machine learning reproducibility [13], [14]. This has been done using Wings/Pegasus provenance workflow [15], and studying provenance differences [16]. Many tools are available to collect provenance information for computation reproducibility such as Reprozip [17], [18]. Beam *et al.* discussed reproducibility challenges for machine learning models in health care [19]. However, repeatability of deep learning models has not yet been discussed as it is typically assumed to hold. Repeatability is a critical requirement for any replicable and reproducible model.

In this article, we provide an overview of deep learning repeatability challenges. More specifically, we discuss deep learning model variations due to uncontrolled randomization. Additionally, we present an experiment showing non-deterministic results when training and re-training U-Net and LeNet-5 with the same exact randomization manual seed value using Keras with TensorFlow backend (version 1.14) [20], [21]. However, we obtained deterministic results using Keras with TensorFlow backend (version 2.1) with LeNet-5 for a classification task, but not with U-Net for a cell segmentation and counting task. Alternatively, we show deterministic models of U-Net and LeNet-5 deep learning architectures trained using Pytorch [22]. The deterministic results were obtained by setting a manual seed for random number generators, and each of Numpy, Torch, and Cuda. Moreover, we show the danger of producing non-repeatable models across deep learning libraries due to the difference in initialization algorithm implementations. Our contributions can be summarized as follows:

- We studied the impact of non-repeatable deep learning models when training with the same data under the same software and hardware settings over multiple trials.
- An experiment with two widely used deep learning development libraries on classification and segmentation tasks was done to investigate deep learning repeatability.
- We provide details of the impact of the weight initialization algorithm implementation differences between Keras and Pytorch.

II. REPEATABIBLITY VS. REPLICABILITY VS. REPRODUCIBLITY

Based on [9], [23], repeatability, replicability and reproducibility are defined as follows:

- *Repeatability*: obtaining the exact same results of an experiment with the same precision by the same team in the same location, under the same experimental setup such as hardware and software settings on multiple trials.
- *Replicability*: obtaining the exact same results of an experiment with the same stated precision by a different team in the same or different location, under the same experimental setup such as hardware and software settings on multiple trials.
- *Reproducibility*: obtaining the exact same results of an experiment with the same stated precision by a different team in a different location, under a different experimental setup such as hardware and software settings over multiple trials.

TABLE 1. Terminology summary.

Repeatability	Replicability	Reproducibility
Same team	Different team	Different team
Same location	Same or different location	Different location
Same experimental setup	Same experimental setup	Different experimental setup

III. REPEATABILITY IMPORTANCE

Deep learning is facing a real repeatability challenge due to uncontrolled randomization. When training deep learning models on GPUs, atomic addition operations can produce variable results. Clearly, if a deep learning model is not repeatable, then the model is not replicable nor reproducible. Controlled randomization refers to variations that can be disabled by setting a manual seed value for tasks such as weight initialization. On the other hand, uncontrolled randomization causes computational variation that can not be disabled for some of development deep learning libraries such as Keras (with TensorFlow). The variation caused by atomic addition is due to the rounding up of floating point numbers and the order of additions as detailed in [24].

In deep learning, initialization, atomic add operations, and reduction approximation during back-propagation (with GPUs) are the leading source of randomization and non-repeatable results with the same software and hardware settings. Although a manual seed value to the random number generators can help fix and control the randomization, other parameters may be uncontrolled in some deep learning libraries such as Keras and TensorFlow. This unique type of uncontrolled randomization plays a significant role in preventing further reliable experimental and algorithmic development in deep learning, as well as making comparisons challenging.

The impact of non-repeatable deep learning models can lead to great frustration. For instance, while designing and tuning a neural network, a deep learning scientist typically tries multiple configurations of hyper-parameters (i.e., learning configurations such as learning rate, batch size, etc.) or even increases the training data by either adding new examples or applying augmentation of the current training examples. Although changing a hyper-parameter value or increasing the number of training examples and obtaining better results is promising, it may not mean the new hyper-parameter or augmentation is good. The difference could be caused by an uncontrolled software randomization during the learning process. Therefore, a repeatable model

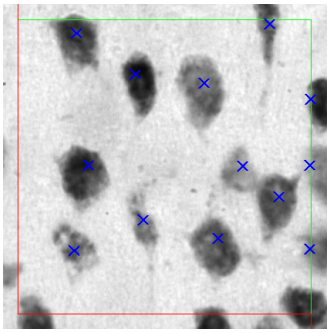


FIGURE 1. Manual annotation where blue X's mark the counted cells. The red lines represent exclusion lines, and the green lines represent inclusion lines.

is critical for efficient parameter selection towards solving performance issues.

A deep learning model with millions of parameters provides best results with quite large data sets for training. The training of large parameter neural networks on large data sets consumes time, energy and computation resources [25]. Therefore, attempts to re-train to check repeatability of such models will increase the cost.

Repeatable deep learning models are significantly important especially in the fields that impact human life directly such as robotics, self-driving cars, and health care.

IV. DATASETS

The datasets for this study are as follows: 1) NeuN single stain dataset of brightfield microscopy images acquired at (100x magnification) of the mouse neocortex. There are a total of 9 mice brain samples as detailed in [26]. The total number of microscopy images is 966. This dataset was used for experimental study of repeatable deep learning model training for segmentation. In particular cell segmentation and unbiased stereology cell counting is done. 2) The MNIST dataset of handwritten digits [27], which we used to study deep learning training repeatability for a classification task.

V. UNBIASED STEREOLOGY

Unbiased stereology is the state-of-the-art for object quantification in stained tissue sections [28]. Unbiased stereology avoids false assumptions that cause systematic errors (bias) by using unbiased systematic-random sampling and unbiased probes. Examples of systematic errors are false assumptions related to shape, size, and orientation of objects [29]. "Here, we use the automatic version of the unbiased fractionator method [30], [31] in which a virtual volume probe disector [32] and unbiased counting rules [33] are used to estimate the total number of NeuN immunostained neurons in sections of mouse neocortex" (Page 1709) [34]. The disector frame is shown in Fig. 1. Cells within or on the inclusion lines (green lines) are counted. However, cells touching the exclusion lines (red lines) are not counted.

VI. EXPERIMENTAL SETUP

The experiment to evaluate deep learning performance variations across training trials was conducted using the U-Net [35] architecture to segment cells in microscopy images, followed by a step for counting cells based on the unbiased stereology approach [34], [36]. For classifying MNIST handwritten digits we used LeNet-5 [27]. The U-Net and LeNet-5 architectures were implemented using both the Keras (backend TensorFlow) and Pytorch deep learning libraries. Both U-Net and LeNet-5 implemented in Keras (backend TensorFlow) were verified to ensure they match the corresponding implementation on Pytorch in terms of convolutional layers, strides, pooling layers, padding, number of kernels per layer, fully connected layers, loss function, initialization, and optimizer settings. Default configurations of Pytorch were changed to match counterpart configurations in Keras such as convolution kernel and bias weight initialization and epsilon (term added to the denominator for numerical stability) of the Adam optimizer [37]. The reason to change Pytorch default configurations to match Keras configurations is to have a fair comparisons between Keras and Pytorch deep learned models results, since there are differences in default configurations including implementation differences of weight initialization as shown in Equations 1 and 2. We trained each deep learning architecture seven times with the same initial seed value for both the Keras (backend TensorFlow) and Pytorch models.

VII. HARDWARE AND SOFTWARE SETTINGS

Training the deep learning models was done on the DeepVision cluster at the University of South Florida. The Deepvision cluster runs the Ubuntu 18.04 operating system and provides eight Graphical Processing Units (GPUs). All GPUs are Geforce GTX 1080ti. The software versions used for training and initialization were as follows:

- Numpy library version 1.17
- Keras deep learning library version 2.3
- TensorFlow deep learning library version 1.14 and version 2.1
- Pytorch deep learning library version 1.3
- Python programming language version 3.6

For training each of the Keras (TensorFlow backend) and Pytorch models, a function was called to set all the seeds needed for randomization to a particular fixed seed value, as described in the documentation of both libraries. The seed value we used was seed $= 2019$. Moreover, we set PYTHONHASHSEED $= 0$ prior to running our code to avoid python hash randomization. The modules we used for seeding randomization algorithms in Keras (TensorFlow 1.14), Keras (TensorFlow 2.1), and Pytorch are shown in Codes 1, 2, and 3 respectively. Each model was cleared from memory before starting training of a new model. Moreover, for training of each model we restricted the training of the deep learning model onto only a single GPU. We tried two versions of TensorFlow (versions 1.14 and 2.1) because the recent release of TensorFlow version 2.1 provides a way to set two important Cuda flags for deterministic results as shown in Code 2. However, TensorFlow version 1.14 does not provide control of Cuda flags.

```
# Keras (TensorFlow backend) setting seeds
    module
def set_seeds(gpu,seed=2019):
    np.random.seed(seed) # numpy random seed
    rn.seed(seed)  # random number generator
        seed
    session_conf = tf.ConfigProto() #
        TensorFlow session configuration
    session_conf = tf.ConfigProto(
            intra_op_parallelism_threads=1,
            inter_op_parallelism_threads=1)
    session_conf.gpu_options.visible_device_list
        = gpu # visible GPU
    from keras import backend as K
    K.set_image_data_format('channels_first')
    tf.set_random_seed(seed) # TensorFlow
        random seed
    sess =
        tf.Session(graph=tf.get_default_graph(),
            config=session_conf)
                #TensorFlow session settings
    K.set_session(sess)
    return sess
```

Code. 1. Keras (TensorFlow 1.14) module for setting manual seed as provided in Keras documentation [20].

```
def set_seeds(gpu,seed=2019):
    np.random.seed(seed) # numpy random seed
    rn.seed(seed) # random number generator
        seed
    session_conf =
        tf.compat.v1.ConfigProto() #
        TensorFlow session configuration
    session_conf = tf.compat.v1.ConfigProto(
                intra_op_parallelism_threads=1,
                inter_op_parallelism_threads=1)
    session_conf.gpu_options.visible_device_list
        = gpu gpu # visible GPU
    os.environ['TF_CUDNN_DETERMINISTIC']
        ='true' #Cudnn deterministic flag
    os.environ['TF_DETERMINISTIC_OPS'] =
        'true' #OPS deterministic flag
    from keras import backend as K
    K.set_image_data_format('channels_first')
    tf.random.set_seed(seed) # TensorFlow
        random seed
    sess = tf.compat.v1.Session(graph=
                tf.compat.v1.get_default_graph(),
                config=session_conf)
                    #TensorFlow session
                    settings
    tf.compat.v1.keras.backend.set_session(sess)
    return sess
```

Code. 2. Keras (TensorFlow 2.1 backend) module for setting manual seed [20].

VIII. EXPERIMENTAL RESULTS

A. SEGMENTATION-BASED UNBIASED STEREOLOGY CELL COUNTING

The dataset used for this experiment is a NeuN single stain data set containing microscopy images of a mouse neocortex [26]. We trained each model on images of 8 mice where there were 845 training images, and we tested on the ninth mouse (test mouse ID was LU3) where the total number of test images was 121. The training set was augmented using a combination of rotation and elastic deformation as detailed in [34]. Each convolutional layer was initialized in Keras using its default convolution initialization algorithm (a modified version of Glorot uniform — sometimes called Xavier uniform) [8]. Glorot uniform is shown in Equation 1 which is an initialization method in Pytorch, and the modified version of Glorot uniform which is used by Keras as the default initialization is shown in Equation 2, where U represents uniform sampling, n_i represents the number of connections from the previous layer, whereas n_{i+1} represents the number of connections to the following layer. So the weights are uniformly randomly sampled from the intervals shown in Equation 1 or 2.

$$K_w \sim U\left[-\sqrt{\frac{6}{n_i+n_{i+1}}}, \sqrt{\frac{6}{n_i+n_{i+1}}}\right] \quad (1)$$

$$K_w \sim U\left[-\sqrt{\frac{6}{\max(1, \frac{n_i+n_{i+1}}{2})}}, \sqrt{\frac{6}{\max(1, \frac{n_i+n_{i+1}}{2})}}\right] \quad (2)$$

The initial weights from Keras were used to initialize Pytorch to avoid any differences in the initialization, because we observed implementation differences as shown in Equations 1 and 2. Each deep learning model was trained for 20 epochs with the Adam learning optimizer, where the learning rate was set to $1e^{-4}$ and l_2 regularization $\lambda = 1e^{-}3$. The results of our experiments show that Keras (TensorFlow version 1.14) is not repeatable due to uncontrolled and non-deterministic Cuda randomization, where the mean error rate and standard deviation of seven trained U-Net models are 7.15% ± 0.737%. Moreover, in Keras (TensorFlow 2.1), where we set the Cuda and TensorFlow deterministic flags: os.environ['TF_CUDNN_DETERMINISTIC'] = 'true' and os.environ['TF_DETERMINISTIC_OPS'] = 'true' (see Code 2), the results were not deterministic (not repeatable), where the mean error rate and standard deviation of seven trained models were 7.14% ± 0.731%. On the other hand, Pytorch showed repeatable results for every run of our model when determinism flags were set (see Code 3). Fig. 2 shows a comparison of Keras (with TensorFlow v1.14 and v2.1) and Pytorch performance when using

```
# Pytorch set seeds module
def set_seed(seed=2019):
    random.seed(seed) # random number
        generator seed
    np.random.seed(seed) # numpy random
        number seed
    torch.manual_seed(seed) # torch seed
    torch.random.manual_seed(seed) # torch
        random seed
    torch.cuda.manual_seed(seed) # cuda seed
    torch.backends.cudnn.deterministic =
        True # cuda deterministic flag
    torch.backends.cudnn.benchmark = False
```

Code. 3. Pytorch module for setting manual seed as provided in Pytorch documentation [22].

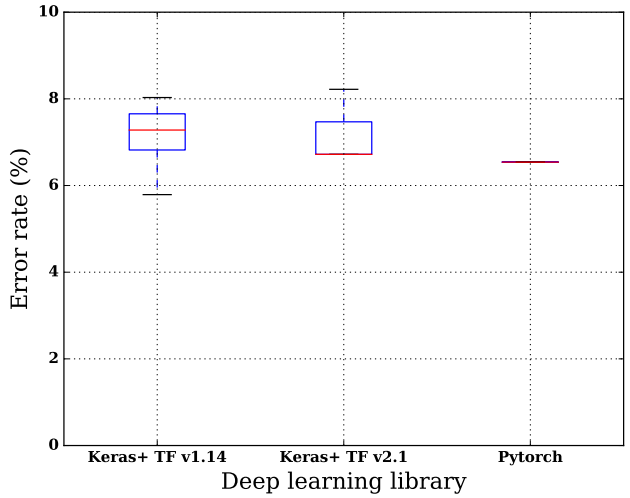


FIGURE 2. Comparison between Keras (TensorFlow (TF) backend) and Pytorch counting results on a test mouse from the NeuN single stain dataset, where Pytorch models were initialized using Keras initial weights (see Equation 2). The plot shows the distribution of the error rates, where the first quartile and third quartile are represented by the blue box, the median is represented by the red line, and the maximum and minimum are shown by the black bars.

TABLE 2. Mean and standard deviation of training and testing time of U-Net trained models using Pytorch vs. Keras+TensorFlow (TF v1.14 and v2.1).

	Keras (with TF v1.14)	Keras(with TF v2.1)	Pytorch
Mean and Std of training time (seconds)	5082.2 ± 19.8	5216.8 ± 24.78	6534 ± 98.4
Mean and Std of testing time (seconds)	7.26 ± 0.80	5.03 ± 0.27	1.99 ± 0.05

the same initialization implemented in Keras (Equation 2). Although, training deep learning models using Pytorch takes more time due to Cuda restrictions that were set, faster testing time is observed for trained models compared to Keras (TensorFlow versions 1.14 and 2.1). A comparison between Keras (with TensorFlow versions 1.14 and 2.1) and Pytorch training and testing time is shown in Table 2. TensorFlow is faster to train and slower to test than Pytorch for this model and data set. Tables 3, 4, and 5 show the error rate, dice coefficient, precision, recall, and F1-score of trained models using Keras (with TensorFlow v1.4), Keras (with TensorFlow v2.1), and Pytorch respectively. Each model was trained seven times. The error rate E and dice coefficient D were calculated using Equations 3 and 4 respectively, where y_t is the actual ground truth count and y_p is the count using deep learning predicted masks. A is the actual ground truth mask, and B is the predicted mask by deep learning.

$$E = \frac{|y_t - y_p|}{y_t} \times 100 \tag{3}$$

$$D = \frac{2 \times |A \bigcap B|}{|A| + |B|} \tag{4}$$

B. CLASSIFICATION OF MNIST HANDWRITTEN DIGITS

The MNIST dataset was used for this experiment, where the training set has 60000 images and the test set has 10000 images. For validation, we randomly selected 10000 images from the training set (i.e., 1000 image per handwritten digit class). We did not apply any augmentation to this dataset. The weights of each convolutional and fully connected layer in Keras were initialized using Equation 2, whereas all bias weights were initialized to zero (i.e., Keras default initialization). For Pytorch, we extracted the initial weights W and bias weight b from the Keras LeNet-5 model to initialize the Pytorch LeNet-5 architecture for training. We trained 7 models of LeNet-5 implemented in Keras (backend TensorFlow versions 1.14 and 2.1) and LeNet-5 architecture implemented in Pytorch. Each model was trained for 20 epochs with the Adam learning optimizer with a learning rate of $1e^{-3}$. The results show that models trained on Keras and TensorFlow version 1.14 are not repeatable, where accuracy mean and standard deviation are $98.99\% \pm 0.044\%$. However, with the deterministic flags available for TensorFlow version 2.1, training deep learning models seven times showed repeatable classification results where

TABLE 3. Segmentation and counting performance of unbiased stereology using U-Net trained models with Keras (TensorFlow version 1.14 backend) for 7 trials under same hardware and software settings.

Model#	Error rate (%)	Dice Coef	Precision	Recall	F1-score
1	7.28	0.856	0.846	0.908	0.876
2	8.03	0.866	0.847	0.915	0.88
3	7.28	0.86	0.846	0.908	0.876
4	6.54	0.871	0.854	0.91	0.881
5	5.79	0.866	0.858	0.908	0.882
6	7.1	0.864	0.848	0.908	0.877
7	8.03	0.863	0.839	0.906	0.871

TABLE 4. Segmentation and counting performance of unbiased stereology using U-Net trained models with Keras (TensorFlow version 2.1 backend) for 7 trials under same hardware and software settings.

Model#	Error rate (%)	Dice Coef	Precision	Recall	F1-score
1	6.72	0.862	0.854	0.912	0.882
2	6.72	0.862	0.854	0.912	0.882
3	6.72	0.862	0.854	0.912	0.882
4	6.72	0.862	0.854	0.912	0.882
5	8.22	0.861	0.839	0.908	0.872
6	6.72	0.862	0.854	0.912	0.882
7	8.22	0.861	0.839	0.908	0.872

TABLE 5. Segmentation and counting performance of unbiased stereology using U-Net models trained with Pytorch for 7 trials under same hardware and software settings.

Model	Error rate (%)	Dice Coef	Precision	Recall	F1-score
1	6.54	0.861	0.856	0.912	0.883
2	6.54	0.861	0.856	0.912	0.883
3	6.54	0.861	0.856	0.912	0.883
4	6.54	0.861	0.856	0.912	0.883
5	6.54	0.861	0.856	0.912	0.883
6	6.54	0.861	0.856	0.912	0.883
7	6.54	0.861	0.856	0.912	0.883

TABLE 6. Mean and standard deviation of training and testing time of trained LeNet-5 models using Pytorch vs. Keras+TensorFlow (TF v1.14 and v2.1).

	Keras (with TF v1.14)	Keras(with TF v2.1)	Pytorch
Mean and Std of training time (seconds)	144.53 ± 4.79	127.54 ± 1.99	222.54 ± 8.62
Mean and Std of testing time (seconds)	4.55 ± 0.13	4 ± 0.18	6.74 ± 0.09

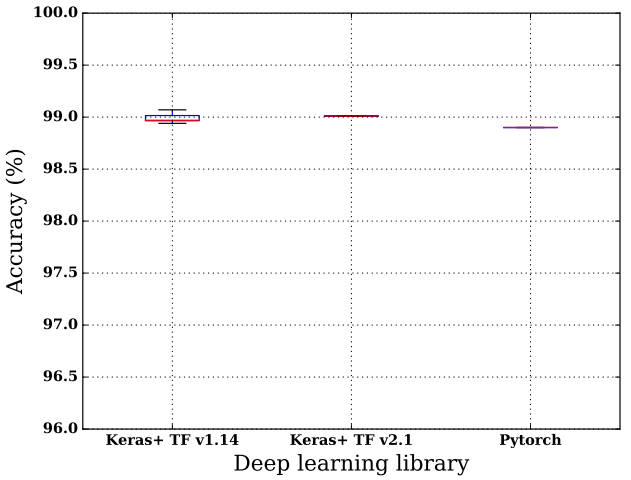


FIGURE 3. Comparison between Keras (backend TensorFlow (TF)) and Pytorch results on MNIST test set, where Pytorch models were initialized using initial Keras weights (see Equation 2). The plot shows the distribution of the accuracy, where the first quartile and third quartile are represented by the blue box, the median is represented by the red line, and the maximum and minimum are shown by the black bars.

TABLE 7. Results of training-retraining of LeNet-5 on MNIST dataset using Keras (backend TensorFlow TF v1.14 and v2.1) and Pytorch for 7 times.

Model	Keras+TensorFlow v1.14		Keras+TensorFlow v2.1		Pytorch	
	Accuracy (%)	FN + FP	Accuracy (%)	FN+FP	Accuracy (%)	FN + FP
1	98.94	106	99.01	99	98.9	110
2	98.96	104	99.01	99	98.9	110
3	99.02	98	99.01	99	98.9	110
4	98.97	103	99.01	99	98.9	110
5	99.07	93	99.01	99	98.9	110
6	99.01	99	99.01	99	98.9	110
7	98.97	103	99.01	99	98.9	110

the accuracy mean and standard deviation are $99.01\% \pm 0\%$. Pytorch based trained deep learning models show repeatable results after setting the Cuda deterministic flag and disabling the benchmark flag (see Code 3). The accuracy mean and standard deviation of seven Pytorch trained models were $98.90\% \pm 0\%$. A visual comparison between Keras (backend TensorFlow v1.14 and v2.1) and Pytorch LeNet-5 models is shown in Fig. 3. In Table 6, we show average training and testing time of Keras (with TensorFlow v1.14 and v2.1) and Pytorch. Training and testing is faster using Keras (with TensorFlow 2.1). Table 7, shows the accuracy of seven independently trained deep learning models using Keras (with TensorFlow 1.14), Keras (with TensorFlow 2.1) and Pytorch.

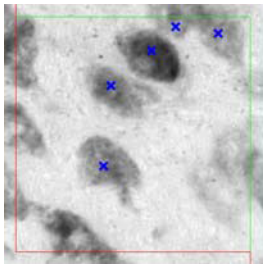
IX. DISCUSSION

Although fixing the same seed for weight initialization in Keras (with TensorFlow 1.14) generates exact initial weight values, a randomization issue occurs during training which is caused by atomic addition operations and reduction during back-propagation. Therefore, when testing models trained multiple times (with the same software configuration including seed value), a varying set of results are obtained. However, training a model in Pytorch generates trained models that are fully repeatable when properly setting the seed values and other important Cuda flags (see Code 3). We have experimented with Pytorch by inverting the deterministic and benchmark flags (see last two lines shown in Code 3), and we obtained non-deterministic results (non-repeatable). However, setting the deterministic flag to true and benchmark optimization flag to false in our Pytorch code give fully deterministic results for both classification and segmentation tasks.

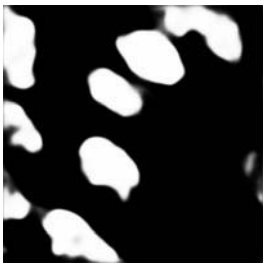
TensorFlow version 2.1 provided a way to set deterministic flags (see Code 2). We obtained repeatable results by setting the deterministic flags for training deep learning models (LeNet-5) to classify MNIST handwritten digits. However, training deep learning model (U-Net) for segmenting and cell counting is not repeatable, and thus we obtained non-deterministic results. Therefore, there are some randomizations in TensorFlow that are not controlled. Fig. 4, shows a manual annotation image (Fig. 4a) and two predicted masks (Fig. 4b and Fig. 4d) using two independently trained U-Net models using Keras (with TensorFlow 2.1). The two predicted masks were post-processed by thresholding at 127 (pixel value), removing small size blobs less than 250 pixels and applying unbiased stereology counting rules by removing cells touching the exclusion line as shown in Fig. 4c and Fig. 4e which shows the difference in the final segmentation and cell counting [26], [36].

It is worth noting that a repeatable result is guaranteed when re-testing a pre-trained model as long as the weights of the trained model are saved. However, if the pre-trained model is trained again with the same hardware and software configurations, it may yield different testing results if the learning process is not repeatable. We also, experimented with testing the trained deep learning models on a different GPU (Nvidia Geforce Titan V) for models previously trained on Nvidia Geforce 1080ti GPU (i.e., training and testing occurs on two different GPUs), and we found that the results are repeatable.

Although, Keras documentation stated that the convolution layers weights initialization is based on the Glorot uniform and provided Equation 1 in the documentation, the actual Keras implementation is different, as shown in Equation 2. This type of inconsistency between documentation and actual implementation is another issue for repeatable, replicable, and reproducible results across different deep learning libraries. When training each of Keras (TensorFlow backend v1.14 and v2.1) and Pytorch based U-Net models with weights initialization based on the Glorot algorithm implemented on each deep learning library (Equation 1 for Pytorch and Equation 2 for Keras), the error rate for an unbiased stereology cell count on the test mouse is shown

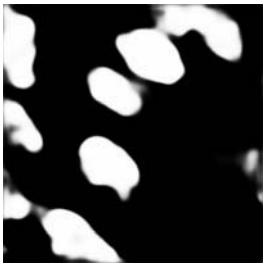


(a) Manual annotation



(b) Predicted mask using model 6

(c) Post-processed mask predicted using model 6



(d) Predicted mask using model 7

(e) Post-processed mask predicted using model 7

FIGURE 4. A manually annotated image is shown in (a) where blue marks represent counted cells based on unbiased stereology counting rules, and inclusion and exclusion of cells is represented by green and red lines respectively. A predicted mask using model 6 (trained using Keras with TensorFlow 2.1) is shown in (b). (c) shows the same predicted mask after post-processing. Another predicted mask using model 7 (trained using Keras with TensorFlow 2.1) is shown in (d) and (e) shows the same predicted mask after post-processing.

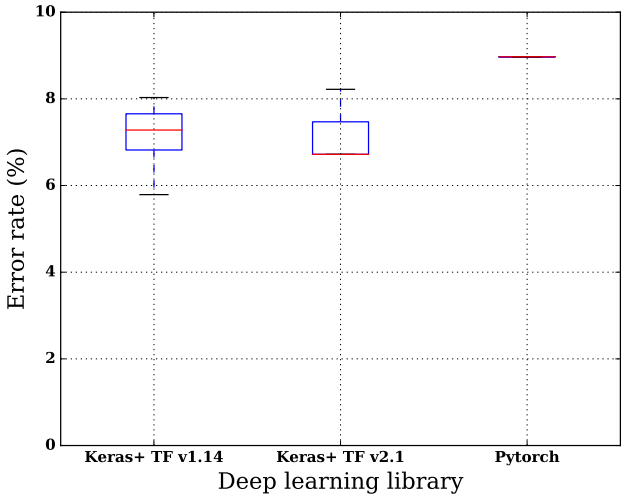


FIGURE 5. Comparison between Keras (with TensorFlow (TF) v1.14 and 2.1) and Pytorch results on a test mouse, where Pytorch models were initialized using the initialization algorithm as shown in Equation 1. The plot shows the distribution of the error rates, where the first quartile and third quartile is represented by the blue box, the median is represented by the red line, and the maximum and minimum is shown by the black bars.

in Fig. 5. In this case, the error rate of Pytorch is higher than Keras (TensorFlow) due to the difference in the weight initialization implementations.

In Active learning approaches [38], K new examples are queried from an unlabeled pool U for the purpose of increasing and diversifying the training set S to obtain a better performing deep learning model M. This approach requires training from scratch or fine tuning a model M in an iterative approach after increasing training set S by K new examples. However, when training deep learning models is not repeatable, different results may be obtained which do not reflect the addition and the usefulness of adding the K queried examples, but instead are caused by model training randomization. Active deep learning and other semi-automatic approaches that require training or fine tuning of a deep learning model frequently after providing new training examples, are highly impacted by the deep learning training repeatability issue.

The performance variations in accuracy across different LeNet-5 deep learning models trained on Keras is very small, however, the confusion matrix shows noticeable variations on the number of cases classified correctly per class. The performance variations in segmentation is larger since the U-Net is classifying each pixel of an image as either part of a cell or background and the counting is done from the generated masks after applying a post-processing step as described in [26], [34].

In this study, we evaluated repeatability of deep learning models for segmentation and classification tasks using the same software and hardware settings. If a model is not repeatable, then it will not be replicable nor reproducible. However, for repeatable models, further evaluation of replicability and reproducibility is required. We did not evaluate the repeatability of training deep learning models across operating systems and deep learning platforms versions. We also, did not experiment with deep learned model repeatability issues in detection and regression problems. Therefore, our future work includes more comprehensive evaluation of detection and regression training repeatability and across platforms and operating systems. Then, as study of replicability and reproducibility of repeatable deep learning models may be done.

X. CONCLUSION

The inability to obtain the same deep learned model when trained from the same initial point multiple times is a problem due to randomization in Keras and TensorFlow version 1.14. However, repeating the same classification and segmentation models is possible using Pytorch deep learning libraries by

fixing the Cuda deterministic flag. Although, TensorFlow version 2.1 provides a way to set Cuda and TensorFlow deterministic flags, the results we obtained were repeatable in the classification of MNIST handwritten digits, but not in segmentation and counting of cells in microscopy images.

There are also problems and variability in implementing certain algorithms across deep learning libraries which creates a hurdle to replicate and reproduce the same trained model using two different deep learning libraries. Some of these implementation differences we discussed are hyperparameters such as epsilon in the Adam optimizer, and the weight initialization implementation using the Glorot algorithm in Keras.

The broader conclusion is that repeatability of deep learning models is important and fundamental for replicable and reproducible deep learning models. Therefore, it has to be checked prior to deploying or publishing a deep learning model. In Pytorch, repeatability when doing deep learning is achievable by fixing all randomizations during training deep learning models such as weight initialization and GPU related randomization for segmentation and classification tasks.

ACKNOWLEDGMENT

Nvidia supported this research with a GPU.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[3] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1145–1153.

[4] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 483–499.

[5] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4724–4732.

[6] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.

[7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[8] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[9] H. E. Plesser, "Reproducibility vs. Replicability: a brief history of a confused terminology," *Frontiers Neuroinform.*, vol. 11, p. 76, Jan. 2018.

[10] X. Bouthillier, C. Laurent, and P. Vincent, "Unreproducible research is reproducible," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 725–734.

[11] C. Drummond, "Replicability is not reproducibility: Nor is it good science," in *Proc. 26th Eval. Methods Mach. Learn. Workshop ICML*, 2009.

[12] P. Thavasimani and P. Missier, "Facilitating reproducible research by investigating computational metadata," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2016, pp. 3045–3051.

[13] S. B. Davidson and J. Freire, "Provenance and scientific workflows: Challenges and opportunities," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1345–1350.

[14] S. Miles, S. C. Wong, W. Fang, P. Groth, K.-P. Zauner, and L. Moreau, "Provenance-based validation of e-science experiments," *J. Web Semantics*, vol. 5, no. 1, pp. 28–38, Mar. 2007.

[15] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar, "Provenance trails in the wings/pegasus system," *Concurrency Comput., Pract. Exper.*, vol. 20, no. 5, pp. 587–597, 2008.

[16] Z. Bao, S. Cohen-Boulakia, S. B. Davidson, A. Eyal, and S. Khanna, "Differencing provenance in scientific workflows," in *Proc. IEEE 25th Int. Conf. Data Eng.*, Mar. 2009, pp. 808–819.

[17] F. Chirigati, D. Shasha, and J. Freire, "ReproZip: Using provenance to support computational reproducibility," in *Proc. 5th USENIX Workshop Theory Pract. Provenance*, 2013, pp. 1–4.

[18] J. F. N. Pimentel, V. Braganholo, L. Murta, and J. Freire, "Collecting and analyzing provenance on interactive notebooks: When IPython meets noWorkflow," in *Proc. 7th USENIX Workshop Theory Pract. Provenance (TaPP)*, 2015, pp. 1–6.

[19] A. L. Beam, A. K. Manrai, and M. Ghassemi, "Challenges to the reproducibility of machine learning models in health care," *JAMA*, vol. 323, no. 4, p. 305, Jan. 2020.

[20] F. Chollet. (2015). *Keras*. [Online]. Available: https://keras.io

[21] M. Abadi (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: https://www.tensorflow.org/

[22] A. Paszke, "Pytorch: An imperative style, high-performance deep learning library," in *Adv. Neural Inf. Process. Syst.*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, Inc., 2019, pp. 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[23] Association for Computing Machinery. (2016). *Artifact Review and Badging*. Accessed: Aug. 31, 2020. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-badging

[24] Pytorch. (2019). *Reproducibility*. Accessed: Aug. 31, 2019. [Online]. Available: https://pytorch.org/docs/stable/notes/randomness.html

[25] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," 2019, *arXiv:1906.02243*. [Online]. Available: http://arxiv.org/abs/1906.02243

[26] S. S. Alahmari, D. Goldgof, L. Hall, H. A. Phoulady, R. H. Patel, and P. R. Mouton, "Automated cell counts on tissue sections by deep learning and unbiased stereology," *J. Chem. Neuroanatomy*, vol. 96, pp. 94–101, Mar. 2019.

[27] 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, 1998.

[28] C. B. Saper, "Any way you cut it: A new journal policy for the use of unbiased counting methods," *J. Comparative Neurol.*, vol. 364, no. 1, p. 5, Jan. 1996.

[29] M. Burke, S. Zangenehpour, P. R. Mouton, and M. Ptito, "Knowing what counts: Unbiased stereology in the non-human primate brain," *J. Visualized Exp.*, no. 27, p. 27, May 2009.

[30] P. R. Mouton, H. A. Phoulady, D. Goldgof, L. O. Hall, M. Gordon, and D. Morgan, "Unbiased estimation of cell number using the automatic optical fractionator," *J. Chem. Neuroanatomy*, vol. 80, pp. A1–A8, Mar. 2017.

[31] M. J. West, L. Slomianka, and H. J. G. Gundersen, "Unbiased stereological estimation of the total number of neurons in the subdivisions of the rat hippocampus using the optical fractionator," *Anatomical Rec.*, vol. 231, no. 4, pp. 482–497, Dec. 1991.

[32] H.-J. G. Gundersen, "Stereology of arbitrary particles* A review of unbiased number and size estimators and the presentation of some new ones, in memory of William R. Thompson," *J. Microsc.*, vol. 143, no. 1, pp. 3–45, 1986.

[33] D. C. Sterio, "The unbiased estimation of number and sizes of arbitrary particles using the disector," *J. Microsc.*, vol. 134, no. 2, pp. 127–136, May 1984.

[34] S. S. Alahmari, D. Goldgof, L. O. Hall, and P. R. Mouton, "Automatic cell counting using active deep learning and unbiased stereology," in *Proc. IEEE Int. Conf. Syst., Man Cybern. (SMC)*, Oct. 2019, pp. 1708–1713.

[35] 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.

[36] P. R. Mouton, *Principles and Practices of Unbiased Stereology: An Introduction for Bioscientists*. Baltimore, MD, USA: Johns Hopkins Univ. Press, 2002.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[38] B. Settles, "Active learning literature survey," Dept. Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep., 2009.

SAEED S. ALAHMARI (Graduate Student Member, IEEE) was born in Abha, Saudi Arabia, in 1988. He received the B.A. degree from King Khalid University, Abha, the M.S. degree from the University of Dayton, Dayton, OH, USA, and the Ph.D. degree in computer science from the University of South Florida, Tampa, FL, USA. He is currently a Faculty Member with the Department of Computer Science, Najran University, Najran, Saudi Arabia. His research interests include medical image analysis, computer vision, machine learning, and deep learning.

DMITRY B. GOLDGOF (Fellow, IEEE) is an educator and scientist working in the area of medical image analysis, image and video processing, computer vision and AI, ethics and computing, bioinformatics, and bioengineering. His research interests are related to two broad thrusts—biomedical image analysis and machine learning with application in MR, CT, PET, and microscopy images, radiomics, and bioinformatics, and video motion analysis with biometrics, surveillance, and biomedical applications. He is currently a Distinguished University Professor and the Vice Chair with the Department of Computer Science and Engineering, University of South Florida, Tampa. He has graduated 29 Ph.D. and 45 M.S. students. He has authored or coauthored over 100 journal and 200 conference papers, 20 books chapters, and edited five books (over 10 000 citations and H-index 52). He is a Fellow of the IAPR, AAAS, and AIMBE. Full CV at http://marathon.cse.usf.edu/ goldgof/.

PETER R. MOUTON received the Ph.D. degree in neurobiology from the University of South Florida (USF), Tampa, FL, USA, and the Karolinska Institute, Stockholm, Sweden. He held a postdoctoral position at the University of Copenhagen, Denmark, and The Johns Hopkins University School of Medicine, Baltimore, MD, USA. He was with the Faculty with the Johns Hopkins Hospital: Pathology and the National Institute on Aging, Baltimore, for 14 years. He is currently a stereologist and neuroscientist with scientific expertise in the development and analysis of human neuropathology in animal models of alzheimer's disease, autism, and down's syndrome. His interest is the development of biotechnology to assist neuroscience researchers. He is the Founder and a current Chief Scientific Officer for SRC Biosciences and a courtesy Faculty with the Department of Computer Science and Engineering, USF. He has authored or coauthored over 100 peer-reviewed books, book chapters, and scientific articles. He is currently a PI on research grants from the National Science Foundation and the National Institutes of Health. He serves as a member of the Editorial Board of the *Journal of Chemical Neuroanatomy*, the National Academy of Inventors, and the Education Committee for the Digital Pathology Association. In 2014, he was named one of Top 12 Technology Innovators in Florida by the *Florida High Technology Magazine*. Since 2016, he has been a Standing Member of an NIH Study Section [Drug Discovery for Aging, Neuropsychiatric and Neurologic Disorders (ETTN-11)].

LAWRENCE O. HALL (Fellow, IEEE) received the B.S. degree in applied mathematics and the Ph.D. degree in computer science in from the Florida Institute of Technology in 1980 and 1986, respectively. He is currently a Distinguished University Professor with the Department of Computer Science and Engineering, University of South Florida. He has authored or coauthored over 85 publications in journals, many conference papers, and book chapters. He has received over $5M in research funding from agencies, such as the National Science Foundation, National Institutes of Health, Department of Energy, DARPA, and NASA. His research interests include learning from big data, distributed machine learning, medical image understanding, bioinformatics, pattern recognition, and integrating AI into image processing. He is a Fellow of the AAAS, AIMBE, and IAPR. He received the Norbert Wiener Award in 2012 and the Joseph Wohl Award in 2017 from the IEEE SMC Society.