Benchmarking Deep Learning Frameworks: Design Considerations, Metrics and Beyond

Ling Liu, Yanzhao Wu, Wenqi Wei, Wenqi Cao, Semih Sahin, Qi Zhang
Georgia Institute of Technology

Abstract—With increasing number of open-source deep learning (DL) software tools made available, benchmarking DL software frameworks and systems is in high demand. This paper presents design considerations, metrics and challenges towards developing an effective benchmark for DL software frameworks and illustrate our observations through a comparative study of three popular DL frameworks: TensorFlow, Caffe, and Torch. First, we show that these deep learning frameworks are optimized with their default configurations settings. However, the default configuration optimized on one specific dataset may not work effectively for other datasets with respect to runtime performance and learning accuracy. Second, the default configuration optimized on a dataset by one DL framework does not work well for another DL framework on the same dataset. Third, we show through experiments that different DL frameworks exhibit different levels of robustness against adversarial examples. Through this study, we conjecture that effectively benchmarking deep learning software frameworks and systems is significantly more challenging than traditional performance-driven benchmarks.

I. INTRODUCTION

Deep learning (DL) applications and systems have blossomed in recent years as more variety of data is entering cyber space and increasing number of open-source deep learning (DL) software frameworks are made available. Benchmarking DL frameworks and systems is in high demand [1], [2], [3], [4], [5], [6], [7], [8], [9]. However, benchmarking deep learning software frameworks and systems is notably more difficult than traditional performance-driven benchmarks. This is simply because big data powered deep learning systems are inherently both computation-intensive and data-intensive, demanding intelligent integration of massive data parallelism and massive computation-parallelism at all levels of a deep learning framework. For instance, a deep learning framework typically has a large set of model parameters and system parameters that need to be configured and tuned. Many of the parameters are interacting with one another in a complex manner from both model learning perspective and system runtime optimization perspective, making the tuning of such large space of parameters substantially more tricky than what have been experimented and understood from systems administration of conventional computer systems, software tools and applications.

This paper presents design considerations, metrics and insights towards benchmarking DL software frameworks through a comparative study of three popular deep learning frameworks: TensorFlow [10], Caffe [11] and Torch [12]. First, we show that although deep learning software frameworks are optimized with their default configuration settings, for a given DL framework, its default configuration optimized to train on one dataset may not work effectively for other datasets. Second, the default configuration optimized on a dataset by one DL framework may not work well when used to train the same dataset by another DL framework. Hence, it may not be meaningful to compare different DL frameworks under the same configuration. Third, different DL frameworks exhibit different levels of robustness in response to adversarial behaviors and different sensitivity boundaries over potential biases or noise levels inherent in different training datasets. Through this experimental study, we show that system runtime performance, learning accuracy, and model robustness against adversarial behaviors and consequence of overfitting are the three sets of metrics that are equally important for effectively configuring, measuring and comparing different deep learning software frameworks.

II. DEEP LEARNING REFERENCE MODEL

Deep learning software frameworks are scalable software implementations of deep neural networks (DNN) on modern computers with many-core CPUs without or with GPUs.

A. DNN Model

A deep neural network refers to a N-layer neural network with N ≥ 2. Learning over the input data to a DNN is typically performed through a sequence of transformations of input data layer by layer with each
layer representing a network of neurons extracted from input data. Although different layers extract different representations of features of the input data, each layer learns to extract more complex, deeper features from its previous layer. A typical layer of a neural network consists of weights \( w \), biases \( b \) and activation function \( \text{act}(\cdot) \), in addition to the set of loosely connected neurons (parameters). It takes as input the neuron map produced from its previous layer, and produces the output as \( y = \text{act}(w \ast x + b) \), where \( x \) is the input of the layer. By connecting these layers together, a deep neural network is a function \( y = F(x) \) in which \( x \in \mathbb{R}^n \) is a \( n \)-dimension input and \( y \in \mathbb{R}^m \) is the output of a \( m \)-dimension vector.

The neural network model \( F \) consists of many model parameters \( \theta_F \). The values of parameters \( \theta_F \) are tuned during the training phase where a large number of input-output pairs as training dataset is fed into the neural network. The training process is conducted in multiple rounds/iterations. During each round, the neural network uses the parameters from the previous iteration with the training data input to predict output forwardly on the N-layer neural network. Then, the DNN computes a loss function between the predicted output and the real (pre-labeled) output. Using the loss function, the DNN updates the parameters using backpropagation with an optimizer, e.g., stochastic gradient descent (SGD) [13] or Adam [14], which minimizes the pre-defined loss function. The general principle of defining a loss function is to measure the difference between the computed output with the ground truth (real output).

In addition to network parameters tuned in the training process, hyperparameters also need to be tuned. The learning rate and batch size are among the most important ones. Both are used to control the extent of parameter updates for each iteration. In general, larger learning rate and/or larger batch size will bring about faster convergence. However, if the learning rate is too large, the training process may not be sophisticated enough and may suffer from fluctuation. If the batch size is too large to fit each mini-batch in memory of GPU or CPU core, the training process may take much longer to complete. Smaller learning rate leads to slower convergence but makes the training process more fine-grained. Smaller batch size leads to larger number of bagging and thus larger number of epochs and may result in lower accuracy, but it can avoid out of memory (OOM) induced runtime performance penalty. Another key hyperparameter is the number of kernels (weight filters), which produces the number of feature maps in each layer of the neural networks. Usually more feature maps enable a deep learning model to give the input data a more refined representation, but at higher runtime cost.

Other hyperparameters, such as the network architecture, number of layers, paddings, strides, kernel sizes of layers, the type of loss function, optimizer, and regularization method, also influence the performance of the deep learning model, each from their own perspective. Note that regularization method can reduce overfitting. Overfitting occurs when the deep learning model is able to achieve high accuracy on the training data but such high accuracy cannot be generalized to the testing data.

Once the DNN is trained, the parameters \( \theta_F \) are fixed, producing a deep learning model that is used in the testing phase to make predictions and classifications. Usually, the trained DNN model may be re-trained before its actual deployment, to ensure that it passes the validation test and the system uses the trained DNN model can provide sufficiently accurate results. The testing phase refers to both the validation and the use of a trained DNN model in real application system.

### B. Reference DL Frameworks

Three mainstream DL frameworks: TensorFlow, Caffe and Torch, are selected for this study.

TensorFlow [10] is an open source software library and implemented based on data flow graph. Neurons are tensors (data arrays) flow between nodes via the edges. The nodes are used to represent mathematical operations and the edges represent the data flow. The tensor based data flow makes TensorFlow an ideal API and implementation tool for Convolutional Neural Networks (CNNs) and Recurrent Neural Neworks (RNNs). However, TensorFlow does not support dynamic input size, which are crucial for applications like NLP.

Caffe [11] supports many different types of deep learning architectures (CNN, RCNN, LSTM and fully connected neural network) geared towards image classification and segmentation. Caffe can be used simply in command lines and it builds neural network models by transforming the data to LMDB format, defining network architecture in the .prototxt file, defining hyperparameters such as learning rate and training epochs in the “solver” file and then training. Caffe works layer-wisely for deep learning applications.

Torch [12] is an open source machine learning library that provides a wide range of algorithms for DL. Its computing framework is based on the Lua programming language [15], a script language. Torch has the most comprehensive set of convolutions, and it supports
temporal convolution with variable input length.

C. Evaluation Metrics

Metrics are a critical component for a benchmark. The following metrics are used in this measurement study.

Training Time. It is a key performance indicator for DL frameworks. Training time is the time spent on building a DNN model over the training dataset. For optimization purposes, models need to be trained for several times with different parameters in order to find the best parameters that achieves the optimal design. Although pre-trained models are made available on many platforms, such as Caffe Model Zoo [16], training is still essential for new model development and for retraining over new datasets or incrementally enhanced datasets.

Testing Time. It is another important performance indicator for DL frameworks. Testing time is the time spent on testing the trained model using a validation dataset. It indicates the potential latency of using the trained model for the prediction or classification based inference when the model is deployed in real-world applications. Thus, testing time affects user experience to a large extent and affects the performance of actual applications. Both training and testing time can be influenced by the configurations of system-specific and model specific parameters.

Learning/Prediction Accuracy. The learning accuracy metric measures the utility of the training framework in the training phase, and the prediction accuracy measures the utility of the trained DNN model at testing phase. Accuracy measurement is highly sensitive to both data-specific parameters, such as the type of datasets, the characteristics of the datasets, such as the number of classes, the number of training samples per class, and the type of deep learning architectures and machine learning library used, such as the collection of algorithms/optimizations included, the configurations of many model specific parameters.

Adversarial Robustness. This metric is designed to measure the resilience of the DL framework and its trained DNN model against adversarial behaviors, including targeted attacks and random (untargeted) attacks, as well as effect of overfitting against potential biases and noise levels in the training dataset during the testing phase. It can also be used a measure to evaluate the effectiveness of regularization techniques deployed in different DL frameworks. For example, TensorFlow uses dropout, while Caffe has weight decay. In this paper, we use the success rate of crafting adversarial examples as a measure of adversarial Robustness.

Two types of adversarial attacks are considered in this study: untargeted Fast Gradient Sign Method (FGSM) [17] and targeted Jacobian-based attacks [18]. Adversarial examples \( x' \) consists of an input \( x \), and its adversarial perturbation \( \delta_x \). With some perturbation, the adversarial example is classified as a new class different from its original class, i.e., \( x' = x + \delta_x \), \( F(x) = y \) and \( F(x') = y' \) both hold while \( y' \neq y \).

FGSM: a simple and fast way to generate untargeted adversarial examples:

\[
x' = x + \epsilon \text{sign} (\nabla_x L(x, y)),
\]

where \( L(x, y) \) is the loss function of original input and the true label. \( \text{sign}() \) is a mathematical function:

\[
\text{sign}(x) = \begin{cases} 
1, & x > 0, \\ 0, & x = 0, \\ -1, & x < 0 
\end{cases}
\]

Jacobian-based attacks: a targeted attack launched by adversaries to generate adversarial examples such that they are classified as targeted class \( t \) instead of their truly legitimate source class. Concretely, for each feature \( i \), the perturbations \( \delta_x \) are formed by decreasing saliency map \( S(x, t)[i] \) of the network rather than the loss function, e.g.,

\[
S(x, t)[i] = \begin{cases} 
0, & \text{if } \frac{\partial F_t}{\partial x_i}(x) < 0 \\
\text{or } \sum_j \frac{\partial F_t}{\partial x_j}(x) > 0, & \left| \sum_i \frac{\partial^2 F_t}{\partial x_i \partial x_i}(x) \right|, \\
\text{otherwise,} & 
\end{cases}
\]

where matrix \( J_F = \{ \frac{\partial F_t}{\partial x_i} \}_{i,j} \) is the Jacobian matrix of the neural network function. The goal of Equation (2) is to reject input features with a negative target derivative or overall positive derivative on classes other than the targeted class. In fact, saliency maps exploit input features that contribute to increasing the probability of target class or decreasing source class or other classes significantly, or both.

III. Experiments

We conduct three sets of experiments to answer the following questions: (i) How effective the default setting (configuration) of one DL framework is in comparison with that of another DL framework for the same datasets? (ii) How efficient the default setting used to train one dataset will be when it is deployed to train another dataset using the same DL framework? (iii) Would the default setting used by one DL framework be effective when used by another DL framework to train the same dataset (i.e., dataset dependent default configuration)? (iv) How well the default setting of a DL framework, which is optimized for training on one dataset, can perform when it is used by another DL framework to train on the same dataset (i.e., framework dependent default configuration)?
All experiments are conducted on Intel Xeon(R) E5-1620 server with CPU: 3.6Ghz, Memory: DDR3 1600 MHz 8GB × 4 (32GB), Hard drive: SSD 256GB, GPU: Nvidia GeForce GTX 1080 Ti (11GB), installed with Ubuntu 16.04 LTS, CUDA 8.0 and cuDNN 6.0.

**DL Frameworks.** TensorFlow [10], Caffe [11], and Torch [12] are selected for this measurement study. Table I shows some of their statistics.

**Datasets.** Datasets play a definitive role in the performance of deep learning in most cases [19], [20], [21], [22]. For the objectives of this study, we choose the two classic datasets: MNIST [23] and CIFAR-10 [24], as all 3 DL frameworks have tuned their configurations for these two datasets. MNIST consists of 70,000 images of ten handwritten digits, each image is 28 × 28 in size. CIFAR-10 consists of 60,000 colorful images of 10 classes, each is 32 × 32 in size.

### A. Default Settings in DL Frameworks

We compare their primary hyperparameter settings used for MNIST and CIFAR-10 in Table II and Table III respectively. All three DL frameworks select their own preferred default training parameters for both datasets. For MNIST, the base learning rate. TensorFlow prefers Adam [14] as its optimizer while Caffe and Torch use SGD [13]. TensorFlow uses the smallest base learning rate, and Caffe uses the largest batch size along with the smallest number of training epochs. Torch uses the largest base learning rate and larger number of training epochs. For MNIST, TensorFlow sets its maximum steps to 20,000, and Caffe sets its max iterations to 10,000, thus by #Epochs = max_steps / #Training Samples, we obtain 20,000 * 50 / 60,000 = 16.67 epochs for TensorFlow and 10,000 * 64 / 60,000 = 10.67 epochs for Caffe. For Torch, the max #Epochs is manually set to 20 and the max_iterations is set to (20 * 60,000) / 10 = 120,000. For CIFAR-10, SGD is used by all as their optimizer. Caffe adopts a two-phase training, the learning rate for its first phase is 0.001 and 0.0001 for the second phase. Also Caffe uses 8 epochs for the first phase training and 2 epochs for the second phase. Using the same formula, TensorFlow has its maximum steps set to (2,560 * 50,000) / 128 = 1,000,000, Caffe sets its max iterations to (10 * 50,000) / 100 = 5,000, and Torch sets it to 20 * 5,000 / 1 = 100,000 for CIFAR-10, since #Training Samples of CIFAR-10 is 50,000 while 60,000 for MNIST.

### Table I: Deep Learning Software Frameworks and Basic Properties

<table>
<thead>
<tr>
<th>Framework</th>
<th>Version</th>
<th>Hash Tag</th>
<th>Library</th>
<th>Interface</th>
<th>Lic.</th>
<th>License</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>1.3.0</td>
<td>ab0ifac</td>
<td>Java, Python, Go, R</td>
<td>1281085</td>
<td>Apache</td>
<td>BSD</td>
<td><a href="https://www.tensorflow.org/">https://www.tensorflow.org/</a></td>
</tr>
<tr>
<td>Caffe</td>
<td>1.0.0</td>
<td>c430690</td>
<td>OpenBLAS &amp; CUDA</td>
<td>Python, Matlab</td>
<td>BSD</td>
<td>BSD</td>
<td><a href="http://caffe.berkeleyvision.org/">http://caffe.berkeleyvision.org/</a></td>
</tr>
<tr>
<td>Torch</td>
<td>torch7</td>
<td>0219027</td>
<td>optim &amp; CUDA</td>
<td>Lua</td>
<td>BSD</td>
<td>BSD</td>
<td><a href="http://torch.ch/">http://torch.ch/</a></td>
</tr>
</tbody>
</table>

### Table II: Default training parameters on MNIST

<table>
<thead>
<tr>
<th>Framework</th>
<th>TensorFlow</th>
<th>Caffe</th>
<th>Torch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Adam</td>
<td>SGD</td>
<td>SGD</td>
</tr>
<tr>
<td>#Epochs</td>
<td>16.67</td>
<td>10.67</td>
<td>20</td>
</tr>
</tbody>
</table>

### Table III: Default training parameters on CIFAR-10

<table>
<thead>
<tr>
<th>Framework</th>
<th>TensorFlow</th>
<th>Caffe</th>
<th>Torch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>SGD</td>
<td>SGD</td>
<td>SGD</td>
</tr>
<tr>
<td>#Epochs</td>
<td>2560</td>
<td>5</td>
<td>20</td>
</tr>
</tbody>
</table>

Next, we compare their primary default parameters about neural network structures, which are configured by the framework creators to work optimally with MNIST and CIFAR10, shown in Table IV and Table V respectively. For MNIST, three frameworks adopt a similar network structure with 2 convolution layers and 2 fully connected layers, derived from LeNet [23]. All set their kernel/neuron size as 5 × 5, 1 → 32 indicates the number of input feature maps is 1. However, other parameters are selected differently for MNIST, such as the activation operation, the number of kernels/feature maps extracted. In comparison, for CIFAR-10, their network structures vary more significantly from the ones configured for MNIST. Caffe and TensorFlow use a 5-layer network structure and Torch uses 4-layer instead. Caffe employs 3 convolution layers and TensorFlow and Torch both use 2 convolution layers.

We make two observations from Table IV and Table V. First, a DL framework may vary their default parameters for different datasets, as shown. We refer to this type of default setting variations as dataset-dependent default settings. Second, different frameworks may NOT use the same default configuration to train the same dataset because each framework may optimize its performance using a different setting of model-turning and system-turning parameters, even when trained over the same dataset. We call this type of default setting variations as framework-dependent default settings, which refers to the default settings used by different DL frameworks to train on a specific dataset. Interesting questions arise:

- Can the default setting optimized to train one dataset be effective to train a different dataset using the same DL software framework? (same framework on different datasets)
B. Impact of Default Settings

We first study the impact of default settings by one framework on different datasets to answer the first question (same framework on different datasets).

This set of experiments compares the performance of three DL frameworks on MNIST using their own default setting optimized for MNIST (Figure 1) and similar comparison also on CIFAR-10 (Figure 2). We highlight three observations: (1) For both datasets, Torch spent longest time in testing as well as its training on MNIST. TensorFlow has the highest accuracy. Even though Torch’s accuracy is higher than Caffe and lower than TensorFlow for MNIST, its accuracy for CIFAR-10 is the worst of the three. One reason that Caffe has shorter training and testing time might be that Caffe was trained for the least number of epochs and less feature maps were extracted for inference, whereas Tensorflow has largest number of feature maps, which helps it to achieve the highest accuracy with low testing time. (2) For MNIST, TensorFlow and Caffe show similar training and testing time, but Caffe has the lowest accuracy. For CIFAR-10, Caffe spent least time for training in CPU or GPU settings and least testing time for GPU setting. TensorFlow has significantly longer training time and its GPU testing time doubles that of Caffe. But TensorFlow has the highest accuracy with Caffe ranking the second, followed by Torch. Note that Torch GPU has slightly lower accuracy (65.61%) than Torch-CPU (66.16%). One reason could be that Torch uses SpatialConvolutionMap [25] on CPU for CIFAR-10, but it lacks the corresponding implementation on GPU. Thus, SpatialConvolutionMM [26] is used as the default. (3) All three frameworks have shortened their training and testing time with GPU for both datasets. Concretely, GPU acceleration for MNIST, TensorFlow is faster by 16 times and 10 times, Caffe is faster by 5 times and 6 times, and Torch is faster by 28 times and 32 times, in training time and testing time, respectively. However, TensorFlow CPU and Torch CPU obtain slightly higher accuracy on MNIST compared to TensorFlow GPU and Torch GPU respectively, though accuracy of all settings on MNIST are above 99% with highest by TF CPU (99.28%) and lowest by Caffe CPU (99.03%). For CIFAR-10, Torch CPU has slightly higher accuracy than Torch GPU. It is worth to note that TensorFlow CPU with 256 epochs can achieve 86.6% accuracy with training time of 21673.81sec (6.02 hours), whereas TensorFlow CPU with 2560 epochs achieves 86.90% accuracy at the cost of training time of 60.88 hours. This result demonstrates that GPU acceleration shortens the training/testing time but it may not ensure high accuracy due to multiple factors, such as mini-batch size, bagging algorithm, per unit memory capacity of GPU. For more detail, see [27].

In summary, the deep learning model demonstrates much better accuracy on the sparse and gray-scale MNIST dataset over the color-rich and content-rich CIFAR-10 dataset. The sparseness and gray scale of MNIST give the data low entropy. We attribute the better

---

**TABLE IV: Primary Default Neural Network Parameters on MNIST**

<table>
<thead>
<tr>
<th>Framework</th>
<th>TensorFlow</th>
<th>Caffe</th>
<th>Torch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Layer (conv)</td>
<td>$5 \times 5, 1 \rightarrow 32$</td>
<td>$5 \times 5, 1 \rightarrow 20$</td>
<td>$5 \times 5, 1 \rightarrow 32$</td>
</tr>
<tr>
<td>ReLU, MaxPooling($2 \times 2$)</td>
<td>MaxPooling($2 \times 2$)</td>
<td>tanh, MaxPooling($3 \times 3$)</td>
<td></td>
</tr>
<tr>
<td>2nd Layer (conv)</td>
<td>$5 \times 5, 32 \rightarrow 64$</td>
<td>$5 \times 5, 20 \rightarrow 30$</td>
<td>$5 \times 5, 32 \rightarrow 64$</td>
</tr>
<tr>
<td>ReLU, MaxPooling($2 \times 2$)</td>
<td>MaxPooling($2 \times 2$)</td>
<td>tanh, MaxPooling($3 \times 3$)</td>
<td></td>
</tr>
<tr>
<td>3rd Layer (fc)</td>
<td>$7 \times 7 \times 64 \rightarrow 1024$</td>
<td>$4 \times 4 \times 64 \rightarrow 500$</td>
<td>$3 \times 3 \times 64 \rightarrow 200$</td>
</tr>
<tr>
<td>ReLU</td>
<td>ReLU</td>
<td>tanh</td>
<td></td>
</tr>
<tr>
<td>4th Layer (fc)</td>
<td>$1024 \rightarrow 10$</td>
<td>$500 \rightarrow 10$</td>
<td>$200 \rightarrow 10$</td>
</tr>
</tbody>
</table>

**TABLE V: Primary Default Neural Network Parameters on CIFAR-10**

- Can the default setting trained on a dataset by one DL framework be used effectively to configure another framework to train the same dataset? (different frameworks on the same dataset)
- Will the default setting, optimized by one framework, work effectively for other DL frameworks? (different frameworks on different datasets)
accuracy performance to the lower entropy of the data since it is easier for the deep learning model to learn. The low entropy also makes the training and testing of the DNN model much faster on MNIST than CIFAR-10.

C. Impact of Dataset-dependent Default Settings

The next set of experiments studies the impact of dataset-dependent default settings and their variations on the performance of different DL frameworks. We configure all three DL frameworks using the same setting on MNIST first, and compare their performance. To be fair, we choose the default setting preferred by each of the three DL frameworks, one at a time. Figure 3 shows the results. In Figure 3a, we compare the training time of the three frameworks on MNIST. There are two colored bars for each framework. For TensorFlow, the training time using its own MNIST default setting is the blue bar and the training time using its own CIFAR-10 default setting is the red bar respectively. Similar evaluation settings for Caffe and for Torch. Testing time comparison is in Figure 3b, and accuracy comparison is in Figure 3c. We make two observations. First, all frameworks perform worst using their own CIFAR-10 default setting on MNIST dataset with longer training and testing time. Recall Table IV and Table V, all three frameworks choose deeper and more complex neural network structures for CIFAR-10, thus, the longer training and testing time. Second, surprisingly, TensorFlow and Torch using their CIFAR-10 default settings on MNIST allow both to achieve high accuracy and almost identical to the best accuracy they produced when using their own MNIST default setting on MNIST. However, Caffe does not enjoy the same result and its performance using its own CIFAR-10 setting on MNIST is surprisingly worsened. In summary, the longer training time and more complex NN structures do not necessarily guarantee higher accuracy. A possible explanation is that over training may bring about the worst overfitting.

Similarly, for CIFAR-10 dataset, Figure 4 shows the comparison results for dataset dependent default settings. Figure 4a and Figure 4b show a similar trend: all three frameworks have much shorter training and testing time when using their default MNIST settings to train and test on the CIFAR-10. This is simply because all frameworks choose simpler neural network structures for training and testing MNIST dataset. Thus, the MNIST default setting runs faster. Now we look at how accuracy responds to the simpler NN structure and shorter training and testing time. Figure 4c shows that both TensorFlow and Caffe suffer from lower accuracy when using
their own default MNIST settings to train and test on the CIFAR-10 dataset, but Torch shows very similar accuracy using either its own MNIST default or its own CIFAR-10 default setting. It is worth noting that Caffe’s performance is very sensitive to its dataset dependent default setting. Thus, for CIFAR-10, when Caffe uses its own MNIST setting, it fails to train on CIFAR-10 such that the training does not converge, resulting in a very low accuracy also at testing phrase. Figure 5 shows the training loss for Caffe as the training process progresses in this experiment. Using Caffe CIFAR-10 default setting on CIFAR-10 dataset, the training loss rate of Caffe during training declines as the training proceeds to 5,000 iterations (expected), while the training loss of using Caffe MNIST default setting on CIFAR-10 is almost constant, at 87.34%, indicating that Caffe using its MNIST setting to train on CIFAR-10 dataset will not converge. Figure 5 measures the training loss by varying the number of iterations since Caffe only provides the statistics for each iteration.

In summary, this two sets of experiments show that for a DL framework, its own default setting optimized for one dataset may not work well for other datasets.

D. Impact of framework-dependent default setting

We evaluate and compare the impact of using different framework default settings when trained on the same dataset. The results show that the default setting that is optimized for training on a dataset by one framework may not work effectively for other frameworks to train on the same dataset.

We first conduct experiments for MNIST. Figure 6 shows the results. In Figure 6a, the first set of three bars compares the training time of using TensorFlow to train on MNIST with three different settings (i) using its own MNIST default setting (blue bar), (ii) using Caffe MNIST default setting (red bar) and (iii) using Torch MNIST default setting (orange bar). Similarly the first set of three bars in Figure 6b shows the testing time of TensorFlow using the three framework-dependent default settings on MNIST, and the first set of three bars in
Figure 6c shows the accuracy of TensorFlow using the three framework settings. The same measurements are done for Caffe and Torch, as shown in the second and third set of 3 bars respectively, in Figure 6. We make two observations. First, using Caffe MNIST default setting, all three frameworks have shorter training time and testing time on MNIST. One reason is that the number of training epochs in Caffe MNIST setting is the smallest, and its NN structure is relatively simple. Second, TensorFlow and Torch achieved the highest accuracy when using their own MNIST default settings to train and test on MNIST, but Caffe has a higher accuracy when using TensorFlow MNIST default setting, at the cost of more than twice the training time. Thus, it is still fair to say that the default setting of Caffe is optimal for MNIST dataset considering both training/testing time and accuracy.

Next, we conduct the same experiments for CIFAR-10 and show results in Figure 7. We highlight three observations. First, Figure 7a shows that the training time of Caffe’s own CIFAR-10 default setting is the shortest as it uses simple NN structures and least number of training epochs. TensorFlow’s own CIFAR-10 default setting works poorly with very long training time when used by Caffe and Torch to train on CIFAR-10 dataset. Second, from Figure 7a and Figure 7b, we observe that Torch’s own CIFAR-10 default setting does not work well when used by TensorFlow on CIFAR-10, because it took much longer training and testing time. The reason can be attributed to the framework-specific implementation and different levels of complexity of their NN structures. Third, TensorFlow and Caffe offer higher accuracy using their own CIFAR-10 default settings. However, Torch achieved much higher accuracy using TensorFlow’s CIFAR-10 default setting than Torch’s own CIFAR-10 default, at a huge cost on training time. Finally, when Caffe is using TensorFlow’s own CIFAR-10 default setting to train on CIFAR-10, Caffe failed to generate a DNN model again as the training process did not converge. The reason is similar to those explained in Section III.C and Figure 5.

Table VI and Table VII provide a summary of the results reported so far on MNIST and CIFAR-10 datasets respectively. Table VIa and Table VIIa show the experimental results with default settings of these three frameworks, serving as the baseline. Table VIb and Table VIIb compared the performance of dataset-dependent default settings, while Table VIc and Table VIIc present the experimental results with framework-dependent default settings.
In this section, we generate adversarial examples in TensorFlow and Caffe on MNIST with their default settings and compare the effective of these adversarial examples in terms of the success rate. We first use untargeted FGSM to launch adversarial attack on neural network (NN) models trained by TensorFlow and Caffe respectively, measure and compare the attack success rates of these models. The parameter $\epsilon$ in the experiment is set to 0.001 (recall Formula (1) in Section II.C).

Figure 8a and Figure 8b show the success rates of the ten digits for TensorFlow trained DNN model and Caffe trained DNN model respectively. It is observed that some digits tend to be crafted more easily into specific classes than to other classes. For instance, consider digit 5, for TensorFlow trained MNIST model, the attack can successfully change its class to digit 3 with highest probability, followed by digit 8, then digit 2 and with small probability to digit 9 and so forth. Similarly, for Caffe trained MNIST model, the FGSM attack has the non-zero probability to misclassify digit 5 to all other 9 classes with different probabilities, but the top 4 highest are the same as TensorFlow: 3, 8, 2, 9. More in-depth analysis can be found in [28].

### Table VIII: Average Crafting Time of Targeted Attacks on MNIST

<table>
<thead>
<tr>
<th>Framework</th>
<th>TF</th>
<th>TF</th>
<th>Caffe</th>
<th>Caffe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>TF</td>
<td>Caffe</td>
<td>TF</td>
<td>Caffe</td>
</tr>
<tr>
<td>average time</td>
<td>113min</td>
<td>92min</td>
<td>187min</td>
<td>134min</td>
</tr>
</tbody>
</table>
Figure 8c compares the average success rate of ten digits by the difference of subtracting the success rate of TensorFlow trained MNIST model (Figure 8a) from the success rate of Caffe trained MNIST model (Figure 8b). We make an interesting observation: in general, the success rate of generating adversarial examples in TensorFlows trained DNN model is lower than that of Caffe. The high success rate for digit 2 shows that digits 2 is the most possible class to which an untargeted adversarial example crafted by FGSM method would fall in.

We then study the impact of Jacobian-based targeted attack on the deep learning models trained on MNIST by TensorFlow and Caffe respectively. To understand whether the size of the feature maps may impact on the attack success rate, for these two sets of experiments, we reduced the feature maps of both at the third layer by the same percentage: for TensorFlow, the feature maps are reduced from 3136 to 1024 and for Caffe, the feature maps are condensed from 800 to 500. We compare the two different sizes of feature maps for both models. It is observed that the smaller number of feature maps tends to result in a much faster rate of crafting adversarial examples, no matter the network model is trained by TensorFlow or Caffe. The result is shown in Table VIII. First, TensorFlow MNIST model is much faster than Caffe MNIST model when all the parameters are the same (either using TF parameter: 113mins v.s. 187mins or Caffe parameter: 92mins v.s. 134mins). Second, the smaller amount of feature maps accelerates the training process even more. Caffe MNIST model with 500 feature maps could generate adversarial examples two times as much as TensorFlow with 1024 feature maps in the same amount of time. For MNIST dataset, the DNN model trained by TensorFlow is to some extend more robust against both types of attacks than the DNN model trained by Caffe. Figure 9 shows the success rate of crafting digit 1 to other nine classes. Table IX shows the success rate result for digit 1 with default regulation methods under different amount of feature maps. The notation TF(TF) means that the TensorFlow framework uses the TensorFlow default parameters, and TF(Caffe) means that the TensorFlow framework uses the Caffe default parameter setting.

We observe that larger number of feature maps, in most cases, would introduce higher robustness regardless of the frameworks. Also, TensorFlow trained model demonstrates higher robustness than that of Caffe. One possible reason is that the dropout in TensorFlow is slightly weaker regularization than the weight decay in Caffe. Such difference may affect the inductive bias of algorithms using one regularizer or the other. Further study on this subject refers to [28], [27].

### IV. RELATED WORK AND CONCLUSION

This paper rethinks the problems of benchmarking deep learning software frameworks.

Several DL benchmark efforts have been put forward [8], [1], [7], [9], [3], [6], [2], [4], [5], each studied a small subset of the popular DL frameworks. However, these proposals do not examine model specific parameters about both neural network structure and hyperparameters of DL frameworks and their interactions with system runtime performance parameters.

We presented a comparative study of TensorFlow, Caffe and Torch with respect to training and testing time, learning and prediction accuracy, as well as model robustness against adversarial examples [29], [30], [18], [31], [32], [33], [17]. We highlight three observations from our in-depth experiments: (1) These deep learning software frameworks are optimized with their default configurations settings. However, the default configuration optimized on one specific dataset may not work effectively for other datasets. (2) The default configuration optimized for one framework to train on a dataset may not work well when used by another DL framework to train on the same dataset. (3) Different DL frameworks exhibit different levels of robustness against adversarial examples. Our study demonstrates that benchmarking deep learning software frameworks is significantly more challenging than traditional performance-driven benchmarks.

### TABLE IX: Impact of Default Feature Maps/Regularization Methods on MNIST

<table>
<thead>
<tr>
<th>Framework</th>
<th>third-layer</th>
<th>Regularization</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF (TF)</td>
<td>3136 → 1024</td>
<td>drop out</td>
<td>0.014</td>
<td>0.802</td>
<td>0.596</td>
<td>0.421</td>
<td>0.022</td>
<td>0.070</td>
<td>0.633</td>
<td>0.991</td>
<td>0.271</td>
</tr>
<tr>
<td>TF (Caffe)</td>
<td>800 → 500</td>
<td>drop out</td>
<td>0.018</td>
<td>0.721</td>
<td>0.482</td>
<td>0.377</td>
<td>0.025</td>
<td>0.112</td>
<td>0.582</td>
<td>0.823</td>
<td>0.119</td>
</tr>
<tr>
<td>Caffe (TF)</td>
<td>3136 → 1024</td>
<td>weight decay</td>
<td>0.594</td>
<td>0.895</td>
<td>0.802</td>
<td>0.721</td>
<td>0.146</td>
<td>0.533</td>
<td>0.912</td>
<td>0.925</td>
<td>0.327</td>
</tr>
<tr>
<td>Caffe (Caffe)</td>
<td>800 → 500</td>
<td>weight decay</td>
<td>0.924</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
<td>0.149</td>
<td>0.870</td>
<td>0.982</td>
<td>0.998</td>
<td>0.441</td>
</tr>
</tbody>
</table>
REFERENCES


