# An Empirical Study of the Impact of Hyperparameter Tuning and Model Optimization on the Performance Properties of Deep Neural Networks

LIZHI LIAO, Concordia University, Canada
HENG LI, Polytechnique Montréal, Canada
WEIYI SHANG, Concordia University, Canada
LEI MA, University of Alberta, Canada

Deep neural network (DNN) models typically have many hyperparameters that can be configured to achieve optimal performance on a particular dataset. Practitioners usually tune the hyperparameters of their DNN models by training a number of trial models with different configurations of the hyperparameters, to find the optimal hyperparameter configuration that maximizes the training accuracy or minimizes the training loss. As such hyperparameter tuning usually focuses on the model accuracy or the loss function, it is not clear and remains under-explored that how the process impacts other performance properties of DNN models, such as inference latency and model size. On the other hand, standard DNN models are often large in size and computing-intensive, prohibiting them from being directly deployed in resource-bounded environments such as mobile devices and Internet of Things (IoT) devices. To tackle this problem, various model optimization techniques (e.g., pruning or quantization) are proposed to make DNN models smaller and less computing-intensive so that they are better suited for resource-bounded environments. However, it is neither clear how the model optimization techniques impact other performance properties of DNN models such as inference latency and battery consumption, nor how the model optimization techniques impact the effect of hyperparameter tuning (i.e., the compounding effect). Therefore, in this paper, we perform a comprehensive study on four representative and widely-adopted DNN models, i.e., *CNN image classification*, *Resnet-50*, *CNN text classification*, and *LSTM sentiment classification*, to investigate how different DNN model hyperparameters affect the standard DNN models, as well as how the hyperparameter tuning combined with model optimization affect the optimized DNN models, in terms of various performance properties (e.g., inference latency or battery consumption). Our empirical results indicate that tuning specific hyperparameters has heterogeneous impact on the performance of DNN models across different models and different performance properties. In particular, although the top tuned DNN models usually have very similar accuracy, they may have significantly different performance in terms of other aspects (e.g., inference latency). We also observe that model optimization has a confounding effect on the impact of hyperparameters on DNN model performance. For example, two sets of hyperparameters may result in standard models with similar performance but their performance may become significantly different after they are optimized and deployed on the mobile device. Our findings highlight that practitioners can benefit from paying attention to a variety of performance properties and the confounding effect of model optimization when tuning and optimizing their DNN models.

CCS Concepts: • **Software and its engineering → Software performance**; • **General and reference** → *Performance*.

Additional Key Words and Phrases: deep neural network, hyperparameter tuning, DNN model optimization, DNN model performance

Authors' addresses: Lizhi Liao, Concordia University, Montréal, Québec, Canada, l_lizhi@encs.concordia.ca; Heng Li, Polytechnique Montréal, Montréal, Québec, Canada, heng.li@polymtl.ca; Weiyi Shang, Concordia University, Montréal, Québec, Canada, shang@encs.concordia.ca; Lei Ma, University of Alberta, Edmonton, Alberta, Canada, ma.lei@acm.org.


Manuscript submitted to ACM

**ACM Reference Format:**
Lizhi Liao, Heng Li, Weiyi Shang, and Lei Ma. 2020. An Empirical Study of the Impact of Hyperparameter Tuning and Model Optimization on the Performance Properties of Deep Neural Networks. 1, 1 (December 2020), 39 pages. https://doi.org/x.x/x.x

## 1 INTRODUCTION

In recent years, deep neural network (DNN) has achieved extraordinary performance[1] over traditional machine learning models in solving many complex problems in various application domains, especially in natural language processing [85], computer vision [51], and gaming [63]. Previous studies show that the effectiveness of intelligent systems often depends on the performance of the DNN model within it. For instance, in autonomous driving, the DNN models are often used to predict the future trajectories to help the autonomous vehicle make reasonable navigation decisions [60]. If the DNN model cannot make predictions in an efficient manner on time, such results may lead to catastrophic field failures. Both the financial and reputational impact of these issues would be detrimental to the success of intelligent systems.

In order to ensure the quality of services provided by the DNN model within the intelligent system, existing solutions mainly focus on two perspectives, which are often applied in combination. The first one is called hyperparameter tuning, which is to search for an optimal combination of model hyperparameters during the training stage of the DNN model. To be specific, DNN models are similar to traditional software systems since they are also highly configurable through providing a set of configuration options for hyperparameters (e.g., loss function or learning rate), and different combinations of hyperparameters may lead to different DNN performance. Therefore, developers often launch a large number of training jobs to systematically explore the best DNN model that is able to meet specific performance requirements (e.g., fast prediction or low energy consumption). The other method, i.e., DNN model optimization, is often performed during the deployment of the DNN-based system. For instance, compared with cloud environments, mobile devices usually have limited computational power, storage, and energy capacity, making it unable to directly deploy a large and complex DNN model on it. Thus, the main focus of the second technique is to compress and optimize existing DNN models to a more compact and smaller model while trying to maintain the performance of the model. As standard DNN models may contain several redundant parameters that can be eliminated, it is possible to prune DNN models by removing such redundancies and shrink the model size while maintaining a similar level of accuracy.

Despite the advantages of the DNN hyperparameter tuning and model optimization methods, practitioners still face numerous challenges when applying these techniques into practice. On one hand, there can be over 12 types of commonly-used hyperparameters in practice [71], each of which contains a wide range of values, resulting in a vast number of combinations of hyperparameters even just for a few types of hyperparameters. For experienced DNN developers, one could determine what types of hyperparameters to tune and their corresponding range of values based on their expertise and experience, while for developers who are unavailable from such knowledge, it is less likely that they will obtain an optimal model configuration since they usually utilize intuition to guide their decisions or they have to choose more hyperparameters and values to tune, making the entire DNN development process expensive and time-consuming. On the other hand, existing optimization techniques mainly focus on shrinking the size of a standard DNN model while keeping its testing accuracy. However, they ignore many other important performance properties, such as inference latency and energy consumption. Changes in such properties may result in many severe problems, making it essential to evaluate them before deployment. For example, suppose two standard DNN models $M_a$ and $M_b$ are obtained after hyperparameter tuning, among which $M_a$ has slightly better performance than $M_b$, i.e., higher

[1]When referring to the general performance of the DNN models, we use the singular form of performance, and when referring to one or multiple specific aspects of the performance (e.g., inference accuracy or inference latency) of the DNN models, we use performance property or performance properties.

inference accuracy and lower inference latency. Then, they are optimized separately and two optimized DNN models $OM_a$ and $OM_b$ are generated, but this time, a different result may appear, i.e., $OM_b$ with similar inference accuracy but much lower inference latency than $OM_a$. In this case, if we simply choose $M_a$ and only apply DNN model optimization on it as it has slightly better performance in terms of the standard model, it may lead entire DNN based systems to suffer from the poor effectiveness of inference. Using such models in the real world scenarios, e.g., in auto-driving vehicles or DNN-based authentication systems, would cause severe field issues and even accidents. All these entail the evaluation of various model performance properties besides model size and accuracy.

Due to the high similarity between DNN models and software systems in terms of high configurability, we regard DNN models as software systems and adopt software engineering analysis techniques to study how tuning different hyperparameters and applying optimization methods affect the performance properties of DNN models in terms of various aspects. We focus on studying several commonly-used hyperparameters in practice and divide them into three different dimensions, including architecture-related hyperparameters, layer-level model training decisions, and optimizer hyperparameters. In terms of model optimization techniques, we apply pruning, quantization, and encoding, as a three-step process, since in previous work [22, 37, 83], these techniques are proven to be quite effective and are often utilized as a combination of optimization techniques. In our effort to provide a more comprehensive study on the effects of tuning different hyperparameters on the standard DNN models for servers/clouds and the optimized DNN models for mobile devices, we try to cover different representative perspectives of the DNN model performance properties, including inference accuracy, inference latency, model size, number of floating-point operations, and battery consumption.

In our study, we use four representative and widely-used DNN models as subject models covering different types of neural networks (i.e., CNN (Convolutional Neural Networks) and RNN (Recurrent Neural Networks)) and various task domains, e.g., image classification, text classification, and LSTM sentiment classification. For each DNN model, we first train it by tuning all studied hyperparameters, then fix each hyperparameter in one dimension at a time, and compare the resulting models with the ones from tuning all the hyperparameters. This is to understand the overall impact of the single hyperparameter. Afterward, we update the list of the best 10 DNN models (based on accuracy) after each hyperparameter tuning and optimize them (the best 10 DNN models) using various techniques, including applying pruning, quantization, and encoding, as a three-step process. Finally, we evaluate all the optimized DNN models for different properties, from which we analyze performance property changes between the standard model and the optimized model and the overall impact of the single hyperparameter on the performance of optimized DNN models. In particular, to understand the impact of tuning different hyperparameters on the performance of the standard DNN models deployed on servers and optimized DNN models deployed on mobile devices, our study aims to answer the following two research questions (RQs):

**RQ1:** *What is the impact of tuning different hyperparameters on the performance of DNN models?*

From our experimental results, we observe that hyperparameter tuning has a significant influence on the different performance properties of the studied DNN models. By examining the impact of tuning different hyperparameters on the DNN model performance in terms of different properties, we find that tuning specific hyperparameters can cause different impacts on the performance of DNN models across models and performance properties. Our findings suggest that practitioners can improve their choice of the tuned models by considering other performance properties while not sacrificing the accuracy of the chosen model.

**RQ2:** *What is the combined impact of hyperparameter tuning and model optimization on the performance of optimized DNN models?*

By applying multiple model optimization techniques on the standard DNN models, we can observe that model optimization may bring obvious differences between the standard DNN models and the optimized ones in terms of various performance properties. Besides, model optimization has a confounding effect on the impact of hyperparameters tuning on the model performance. Our findings imply the importance of considering the impact of subsequent model optimization when building and tuning DNN models in the cloud/server environments.

The empirical study results of our work highlight the impact of various hyperparameters tuning on the performance of both standard and optimized DNN models, and have the following key implications:

- As current hyperparameter tuning tools (e.g., Keras Tuner [66] or Hyperopt [12]) are mainly based on tuning for one objective (e.g., accuracy or loss), practitioners need to take other specific performance properties into account when conducting hyperparameter tuning and do not always choose the top-1 DNN model from hyperparameter tuning as the final decision.
- The performance characteristics of standard DNN models for servers/clouds may be different from that of the corresponding optimized DNN models for mobile platforms, thus practitioners need to be careful not to simply transfer the hyperparameter configurations or the understanding of the impact of such hyperparameter settings from one platform to another platform (e.g., tuning hyperparameters on the server and transferring them to mobiles). Instead, one needs to consider performing hyperparameter tuning on the target devices for deployment.
- There exist interactions among multiple hyperparameters (especially the hyperparameters within the same dimension). These different hyperparameters often influence each other and their impact varies across different DNN models and performance properties. Thus, for practitioners, specific considerations about what hyperparameters to tune are required in the context of specific DNN models and performance requirements.

Our findings also provide insights for practitioners who are interested in DNN hyperparameter tuning and model optimizing in order to achieve specific performance requirements.

**Paper organization.** Section 2 discusses the background and related work of our study. Section 3 outlines the setup of our case study on four subject DNN models. Section 4 discusses the results by answering our two research questions. Sections 5 discusses the implications based on the results. Section 6 presents the threats to the validity of our findings. Finally, Section 7 concludes the paper.

## 2 BACKGROUND AND RELATED WORK

In this section, we first introduce the background of our study, including the performance of DNN models, hyperparameter tuning, and representative DNN model optimization techniques. Then, we discuss previous work relevant to this paper along two directions, i.e., DNN performance and DNN optimization.

### 2.1 Background

*2.1.1 Performance of DNN models.* A DNN model is essentially a collection of mathematical functions that are structured by deep learning frameworks (e.g., TensorFlow or PyTorch) as tensor-oriented computation graphs. Such DNN graphs have many important quality attributes (i.e., non-functional properties) that can be used as measurements for the algorithm, structure, and complexity of a DNN model. Although a DNN model can be evaluated from a variety of perspectives, many performance properties may have a strong correlation with one or few of other ones, for instance, the energy consumption of a DNN often has a linear relationship with the model efficiency that is defined as the number of inference it can make per second. Therefore, in our study, we consider multiple representative properties,

i.e., inference accuracy and latency, model size, number of floating-point operations (FLOPs), and battery consumption, as DNN model performance indicators.

*2.1.2 DNN hyperparameter tuning.* A typical DNN model often has two categories of parameters. The first type is trainable parameters, which can only be learned by the training process, for instance, the weights of a neural network are trainable parameters. The second type is hyperparameters, e.g., the number of units in a dense layer and the learning rate, which need to be set before launching the training process [46]. Similar to the traditional software system, a DNN model is often highly configurable by providing numerous configuration options for hyperparameters, and even for small DNN models, tuning these hyperparameters can be computationally expensive. Nevertheless, hyperparameter tuning is very important as an optimal combination of hyperparameters may lead to significant improvement in the performance of a DNN model [86]. Therefore, in order to search for the optimal model configuration meeting specific development requirements, practitioners often leverage open-source libraries (e.g., Hyperopt or Keras Tuner) and apply a variety of hyperparameter search algorithms (e.g., bayesian optimization or random search) to automatically perform large numbers of trials to train DNN models with a diversity of hyperparameter configurations.

In particular, depending on the mechanisms, the existing hyperparameter optimization tools can be divided into two types: 1) fully automatic optimization tools and 2) semi-automatic optimization tools. There are multiple prevalent fully automatic optimization libraries to cater to the demand for DNN hyperparameter tuning. For example, AutoKeras [45], Auto-WEKA [49, 82], Auto-sklearn [32], and Google Cloud AutoML [15] are the implementations for automated machine learning (AutoML) which automate the tasks covering the complete pipeline from the raw dataset to the deployable DNN models. They also provide functions to automatically search for architecture and hyperparameters of DNN models without the need for any expert knowledge about the DNN models and techniques. Whereas for the second type (i.e., semi-automatic optimization tools), for example, Keras Tuner [66], Hyperopt [12], and HpBandSter [30], developers need first to construct the DNN model to be tuned and then configure the search scope of the hyperparameters, then the tools can automate the process of selecting the right set of hyperparameters from the search scope for the DNN model. Comparing to the first type (i.e., fully automatic optimization tools), the second type requires more effort in defining the DNN model and the hyperparameter search scope, however, it is more flexible and has better capability to handle various tasks and inputs, while the first type mainly focuses on processing text [15] and image [29] related tasks. In this work, we opt to use Keras Tuner as the hyperparameter optimization tool to pick the optimal set of hyperparameter combinations for DNN models since it provides full flexibility and convenient APIs to define the hyperparameter search space. Besides, it is easy to leverage the included algorithms to find the best hyperparameter values. Keras Tuner comes with built-in Bayesian Optimization, Hyperband, and Random Search algorithms, and also allows researchers and practitioners to experiment with their own search algorithms.

*2.1.3 DNN optimization.* Over the past few years, deep learning has achieved great success in numerous application areas, e.g., natural language processing (NLP) or computer vision (CV). However, existing DNN models are often computationally expensive in many aspects, e.g., computational requirement and model size, which hindering the deployment on resource-constrained devices (e.g., mobile devices or IoT devices). Therefore, many studies have recently been conducted in optimizing DNN models without significantly decreasing the model performance (e.g., accuracy) [21]. In this paper, we briefly review the three most commonly adopted DNN model optimization techniques in practice [6, 22, 37, 83, 89] and they are separately pruning, quantization, and encoding.

**Pruning.** Pruning in deep neural networks has been taken as an idea from synaptic pruning that happens in the human brain. Synaptic pruning is a natural process that occurs between early childhood and adulthood, and during such

process, axon and dendrite gradually decay and eventually die off resulting in synapse elimination. Inspired by such a common biological phenomenon, a DNN model can also be pruned in a similar way. In deep learning, DNN model pruning can be performed at different granularities to get a neural network with a smaller size. The first type is weight pruning which removes redundant connections between two neurons present in the DNN architecture and it often involves cutting out unimportant weights which are usually defined as weights with small absolute values. While the second type is performed at the neuron level, i.e., neuron pruning, where an entire neuron or several neurons with all the related weights in the neural network are deleted. In practice, DNN model pruning often requires to re-train or fine-tune the neural network to regain the accuracy, since the model was actually trained for the original connections and the model accuracy is probably to drop after removing some neurons or weights. We would like to point out that dropout [76] seems to be partially similar to pruning since they both omit some neurons and their connections in the DNN models. However, dropout and pruning have different purposes and mechanisms. Dropout is a regularization technique used to prevent overfitting during the training phase. During dropout, neurons are randomly selected and ignored, and those ignored neurons are temporarily removed on the forward pass, and their weights are not updated on the backward pass. For pruning, it can also serve as a mechanism to avoid overfitting since it also removes a part of the network, thereby reducing the complexity of the model and limiting the risk of overfitting. However, its main purpose is to optimize the model in order to provide the model with more efficiency (i.e., faster inference and smaller model size), and it can be conducted at both the model training phase and optimization phase. During pruning, unimportant neurons or weights (i.e., providing little predictive power for the problem) determined by the algorithm are permanently removed and only the important neurons and weights remain. Therefore, in our work, we only consider the pruning in our DNN model optimization pipeline. Specifically, we opt to leverage the pruning approach implemented in the TensorFlow Model Optimization Toolkit [4] which performs fine-grained magnitude-based weight level pruning to optimize the standard DNN models to generate optimally-sized models.

**Quantization.** Neural network model quantization is based on a fundamental idea of replacing high precision floating point format (e.g., 64-bit or 32-bit floating point) with low precision floating or even fixed point format (e.g., 16-bit floating point or 8-bit fixed point integer) to reduce the number of bits required for storing the weights of neural networks and consequently compress the standard DNN model. For example, if a DNN model with a 32-bit floating point format is converted to the 8-bit fixed point integer format, the new DNN model can save up to 75% of the number of bits for storing one weight comparing to the standard model. At present, there are mainly two forms of quantization, the first kind involves bundling weights together by clustering them and thus using fewer distinct float values to represent more weights, and the second quantization technique is converting high precision floating point weights to low precision floating or fixed point representation by rounding them off. In addition, such quantization techniques can be performed both during the time of training a DNN model and on an already-trained high precision floating point DNN model.

In this study, we opt to use the post-training INT8 quantization which is the most commonly adopted quantization strategy in practice [24, 78, 88]. In particular, the standard DNN model is trained in 32-bit floating point format, while during inferencing, the most critically intensive parts are computed with 8 bits instead of floating points. Besides, we directly utilize the TensorFlow official optimization libraries, i.e., the TensorFlow Model Optimization Toolkit [4], which by default provides the implementation for various quantization methods.

**Encoding.** A trained DNN model is basically a file including the layers and weights in the DNN, which are often exported or saved in a binary file format that can potentially be compressed. As an inseparable part of file compression techniques, encoding plays a vital role in exploring the data file content to find redundancy and patterns that allow

for abbreviating the content in a way to take up less space yet maintain the ability to reconstruct a full version of the standard when needed [92]. In general, encoding techniques are divided into two main categories. One is fixed-length encoding, e.g., ASCII, where each character is stored in the same amount of space, and such methods are convenient because the boundaries between characters are easily determined and the pattern used for each character is completely fixed. Another type is variable-length encoding, e.g., Huffman encoding, where some characters may only require fewer bits while other characters may require more bits. One major benefit of such an approach is that it requires less space overall, since it considers the occurrence probability for each symbol and represents more common symbols with fewer bits instead of full fixed-length bits.

As a typical example of variable-length encoding, Huffman encoding has been widely adopted in practice for DNN model compression as it has several advantages [37]. First, it is a lossless data compression approach that can prevent losing any important information when compressing a DNN model. Second, it uses an optimal prefix-free coding and can take advantage of skewed or biased distributions of the DNN model's effective weights to further compress a DNN model by representing more common weights with fewer bits. Based on these benefits, Huffman encoding is adopted after pruning and quantization in the experiments of our work to provide further compression of the DNN models.

### 2.2 Related work

*2.2.1 Studies on performance of DNN.* Several studies were conducted to gain a deep understanding and assure the performance of deep learning models in terms of different aspects. To comprehend the performance of DNN models, Li et al. [53] conducted a detailed quantitative characteristic study on the power behavior and energy efficiency of various prevalent convolutional neural network (CNN) models during training and predicting time, which includes fine-grained analysis comparing different neural network layer types under both CPU and GPU platforms. Canziani et al. [19] performed a comprehensive study on analyzing the relationships among multiple important properties in practical DNN applications, including accuracy, memory footprint, parameters, operations count, inference time, and power consumption, providing insights into designing an efficient DNN model. Pei et al. [68] first proposed the concept of neuron coverage and also generated thousands of counterexamples which can help practitioners build robust DNN models. Prior research illustrates the importance of comprehending and ensuring the DNN performance in practice. In comparison, this paper focuses on studying how tuning different hyperparameters impact various aspects of performance in DNN models. Our work can be adopted in practice in tandem with the prior research on the topic of comprehending DNN performance .

*2.2.2 Studies on the impact of hyperparameter tuning.* Given the importance of DNN performance, tuning hyperparameters is commonly-adopted in practice to ensure performance when constructing a DNN model. Wong et al. [86] performed a case study on comparing the performance of machine learning models with and without tuning hyperparameters and their result shows the effectiveness of tuning hyperparameters for the assurance of model performance. In order to assess the influence of hyperparameters, prior work [31, 42] proposes analysis techniques that identify the hyperparameters that contribute most to the model performance improvement after tuning. Based on that, Hutter et al. [43] propose a more general approach, which can quantify the relative importance of both single hyperparameters and the interactions between hyperparameters by using random forest model predictions within a functional ANOVA framework [34]. However, these works mainly focus on the limited performance properties (e.g., inference accuracy) and device (e.g., server), while we study a variety of performance properties (e.g., inference accuracy, inference latency, model size, FLOPs, and battery consumption) and conduct experiments on two different platforms (e.g., server and

mobile devices). Our main focus is to generalize the implications and rules of thumb to different DNN models and help to prioritize which hyperparameter choices in order to facilitate a better general understanding of hyperparameter effects and better decision-making for future experiments. In addition, neural architecture search (NAS) that aims to automatically design a DNN architecture to achieve optimal performance on a certain task has attracted a lot of attention in the past few years. There are several prior studies [10, 17, 56, 72, 81] on studying NAS and DNN model performance, however, compared to NAS that automatically design the whole model, our study focuses on the approach that gives developers more freedom to design the DNN structures and only automate the process of tuning the hyperparameters to acquire an optimal model. Such a way is more flexible and has the better capability to handle various tasks and inputs (e.g., graph data) in different usage scenarios and contexts, while NAS mainly focuses on computer vision and NLP related tasks [62, 70]. Thus, we would leave NAS as our direction for future work. Besides, traditional brute-force grid search for hyperparameters is expensive as it often results in numerous combinations even for a small number of hyperparameters[11]. Thus, to make the whole process affordable, several efficient alternative hyperparameter search algorithms, e.g. random or Bayesian hyperparameter search techniques, were proposed in prior work [11, 13, 75].

*2.2.3 Studies on optimization of DNN.* Due to the complexity of the deep learning models and the required resources, in recent years, extensive research has been conducted in order to investigate and design optimization techniques to accelerate DNN model deployment and execution on mobile devices. Han et al. [37], Tung et al. [83], and Choi et al. [22] conducted experiments combining pruning, quantization, and Huffman encoding into a three-stage pipeline that significantly reduces the size of DNN models with minimal loss of accuracy. Li et al. [54], Pavlo et al. [64], Ayinde and Zurada [8], and He et al. [40] proposed an acceleration approach to effectively prune filters with low weight magnitudes and unimportant parameters in deep and/or wide CNN models, and thus improve the resource efficiency (i.e., reduce DNN computation cost (FLOPs)), making it possible to deploy and execute the complex DNN models on embedded sensors or mobile devices where computational and power resources may be limited. In addition to the model size and energy efficiency, Kim et al. [47] also proposed a CNN compression scheme in order to obtain significant reduction in wall-clock time used when executing DNN models on several mobile devices. Different from prior work, we conduct a comprehensive study on multiple performance aspects of DNN models with a combination of multiple optimization techniques (i.e., pruning, quantization, and encoding) and hyperparameter tuning applied and compare their performance with the standard DNN models. Therefore, our empirical studies and findings in this paper complement existing work in order to help practitioners gain better understanding of how the state-of-the-art model optimization techniques and tuning different hyperparameters affect different DNN model performance properties.

## 3 EMPIRICAL STUDY SETUP

To study the impact of tuning different hyperparameters on the performance of standard DNN models and optimized DNN models for mobile devices, we perform case studies on four prevalent DNN models in use as well as three large and classical datasets[2]. In this section, we first present the design of our empirical study. Then the subject models and the datasets used for each model are described. Afterwards, we introduce the hyperparameters and their ranges selected in our study to tune the subject models, and the properties of the DNN models that we focus on in our work. Finally, we present the hardware and platforms on which our experiments are performed.

[2]Our experimental setup, scripts and, results are shared online https://github.com/senseconcordia/TOSEM2021Data as a replication package

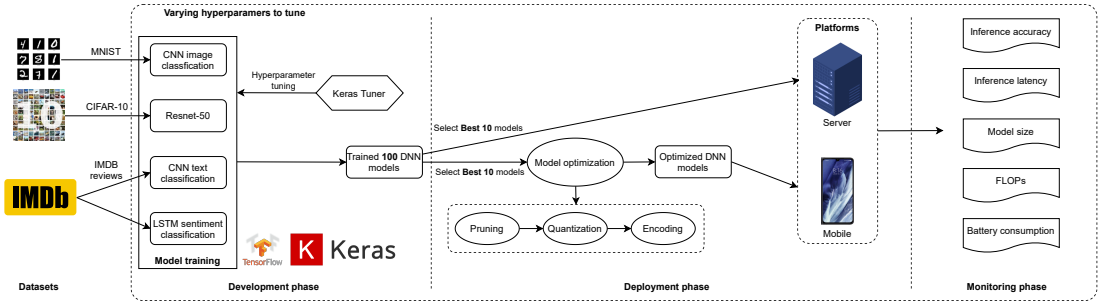



Fig. 1. An overview of our empirical study

### 3.1 Study design

Figure 1 shows the overview of our empirical study, which contains three main phases: 1) development phases, 2) deployment phase, and 3) monitoring phase. In the development phase, we select 4 widely-used DNN models covering both CNN and RNN architectures, i.e., *CNN image classification*, *Resnet-50*, *CNN text classification*, and *LSTM sentiment classification*, from the official TensorFlow website [2], as our empirical study subjects. We use 3 publicly available datasets (i.e., *MNIST*, *CIFAR-10*, and *IMDB reviews*) for the training and testing of the subject DNN models. We also utilize *Keras Tuner* to automatically train the DNN models using 100 combinations of hyperparameters from a large hyperparameter search space for each of the four types of model. In the deployment phase, since our work focuses on studying the impact of tuning different hyperparameters on the performance of DNN models deployed across various hardware platforms, we have selected two popular platforms to deploy and evaluate the DNN model's performance, including the Linux server and the Android mobile device. In addition to deploying all 100 trained DNN models on the server platform, we also have selected the best 10 DNN models (based on the training accuracy) and applied various optimization techniques (i.e., pruning, quantization, and encoding) to make them suitable for deployment on the mobile platform. After the deployment of these DNN models on the various platforms (i.e., server and mobile), in the monitoring phase, we measure the performance properties of each DNN model, including inference accuracy, inference latency, model size on the disk, and floating point operations (FLOPs). For the mobile platform, we also measure battery consumption which is critical for mobile applications. It is worth noting that these three phases are repeated every time we tune the varying hyperparameters in order to study the impact of hyperparameter tuning and model optimization on the performance properties of DNN models.

### 3.2 Subject models and datasets

In our experiments, we choose four representative and widely-used DNN models as our subject models, namely *CNN image classification*, *Resnet-50*, *CNN text classification*, and *LSTM sentiment classification*. All of them are from the official TensorFlow website [2]. These subject models cover different types of neural networks, including CNN (Convolutional Neural Networks), RNN (Recurrent Neural Networks) and various domains, including computer vision and natural language processing. The details of each subject DNN model and corresponding datasets are shown in Table 1.

We use three classical datasets in the deep learning domain to perform our experiments: *MNIST* [52], *CIFAR-10* [50], and *IMDB reviews* [61]. For the text classification tasks (i.e., *CNN text classification* and *LSTM sentiment classification*), the *IMDB reviews* dataset is adopted. *IMDB reviews* is a large movie review dataset for binary sentiment classification,

including a set of 25,000 highly polar movie reviews for training, and 25,000 for testing. For the image classification tasks (i.e., *CNN image classification* and *Resnet-50*), we apply two different datasets including *CIFAR-10* and *MNIST* in our experiments. *MNIST* has 70,000 $28 \times 28$ gray-scale images including 60,000 training and 10,000 testing samples of handwritten digits and its output labels are the 10 numbers from 0 to 9. *CIFAR-10* consists of 60,000 $32 \times 32$ colour images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 testing images. It is worth noting that since *Resnet-50* has a more complex structure comparing to *CNN image classification* (i.e., 50 layers in the *Resnet-50* model, while only 6 layers in the *CNN image classification* model), so *Resnet-50* is likely to perform better on more sophisticated images (i.e., classifying 10 classes of real-world objects in *CIFAR-10*) than *CNN image classification*. While for *CNN image classification* with a less complicated and more compact structure, the relatively straightforward classification workload (i.e., identifying the handwritten number from 0 to 9 in *MNIST*) would be more appropriate. In addition, due to the limitation of model energy consumption and mobile device's battery capacity, for *LSTM sentiment classification*, we use all the 25,000 training samples for training, while only apply the first 1,000 testing samples for inference.

Table 1. Overview of our subject DNN models and datasets

| Model | Dataset | # Training samples | # Testing samples |
|---|---|---|---|
| CNN image classification | MNIST | 60,000 | 10,000 |
| Resnet-50 | CIFAR-10 | 50,000 | 10,000 |
| CNN text classification | IMDB reviews | 25,000 | 25,000 |
| LSTM sentiment classification | IMDB reviews | 25,000 | 1,000 |

### 3.3 Hyperparameters of DNN models

Hyperparameters are variables that we need to set before applying a machine learning algorithm to a dataset. In general, there are no magic numbers that can work for all cases and the optimal values often depend on the specific task and dataset [1]. To understand the impact of hyperparameter tuning on the performance of the DNN models and optimized DNN models, we structure our experiments along with some commonly-used hyperparameters in practice into three dimensions, including architecture-related hyperparameters, layer-level model training decisions, and optimizer hyperparameters. The description and search space of each hyperparameter used in our subject DNN models are shown in Table 2. It is worth noting that for the embedding dimensions hyperparameter which defines the size of the vector space of which each word will be embedded, we only consider this hyperparameter in text classification tasks (i.e., *CNN text classification* and *LSTM sentiment classification*). Although one could regard the dimension of the input layer of the image classification tasks (i.e., *CNN image classification* and *Resnet-50*) as an embedding layer, altering the input size of the images is not considered in our study as the size of the images are fixed. Hence, we consider the embedding dimensions for image classification tasks as not applicable (i.e., marked as n/a). Regarding the kernel size and the pooling method hyperparameters, since we do not have any kernels and pooling layers in the *LSTM sentiment classification* model, the kernel size and pooling method hyperparameters would not be applicable (i.e., marked as n/a) in this case.

We opt to use a popular hyperparameter tuning toolkit called *Keras Tuner* [66] to automatically search for the optimal DNN hyperparameter combinations from the large hyperparameter search space covering over 11 types of hyperparameters, each of which contains a wide range of values. In order to adopt *Keras Tuner* to conduct hyperparameter

tuning, we first need to design the DNN model architecture for hyperparameter tuning and define the hyperparameter search space. Then, a hyperparameter search algorithm (e.g., Bayesian optimization or random search) needs to be selected to instantiate the hyperparameter tuner, and meanwhile, an objective of whether to minimize model loss or maximize the model accuracy needs to be specified to optimize the DNN model. Afterwards, *Keras Tuner* will automatically search for the optimal hyperparameters and train the DNN models. Finally, after hyperparameter tuning is finished, a set of DNN models with various hyperparameter configurations and different performance properties (e.g., inference accuracy or model size) will be generated.

Search in such a large space (with the magnitude of $10^{10}$ resulting from configuring 11 types of hyperparameters with multiple values) is challenging. Compared to naive grid search, random search is more efficient and allows trials in a larger hyperparameter search space. Compared to Bayesian Optimization and Hyperband algorithms, random search has less complex technicalities and can achieve comparable performance. In particular, as proven in prior research [11], random search is able to find better models by effectively searching through a larger configuration space with the same computational budget. For example, if the close-to-optimal region of hyperparameters occupies at only 5% of the grid surface, then the probability that all of them miss the desired interval of 5% is $(1 - 0.05)^n$, where n is the number of trials. So the probability that at least one of them succeeds in hitting the interval is $1 - (1 - 0.05)^n$, which means that random search with 100 trials will find that region with a 99% probability. Thus, we apply the random search algorithm that is implemented in *Keras Tuner* to randomly sample 100 trials of hyperparameter combinations. It should be noted that, during DNN model hyperparameter tuning, we first determine a conditional scope of the hyperparameter values to make sure that there are no invalid hyperparameter values for each individual parameter. If there are invalid combinations of hyperparameters when building the model, the generated model could be either invalid or introduce a very poor performance (e.g., low accuracy). In such cases, our approach will continue to try other combinations until reaching the desired number of valid combinations of hyperparameters. Besides, we only choose the best 10 DNN models from the resulting 100 models for comparison and perform further optimization as these models achieve the best performance on the objective metrics (e.g., accuracy or loss) and are more likely to be selected as the final model than the rest of the models (cf. Section 4).

### 3.4 DNN model performance properties

In order to investigate the effects of tuning different hyperparameters on the standard DNN models for server or cloud platforms and the optimized DNN models for mobile devices, we focus on the evaluation of the DNN models on different representative perspectives, including inference accuracy, inference latency, model size, number of floating-point operations, and battery consumption. In the following, we briefly introduce each of them and describe how we measure them in our study.

**Inference accuracy.** The inference accuracy is a critical perspective of DNN model performance as it directly determines the quality of predictions of the DNN model and those predictions further affect the scientific evidence for making decisions. The higher the inference accuracy that one DNN model can achieve, the more promising that model can be in practice. The DNN model inference accuracy can be measured by the number of correctly predicted data samples divided by the total number of data samples in the testing dataset. It is worth noting that in our experiments, the datasets are balanced to avoid any biases in the inference accuracy and reflect the real forecasting power of the DNN model.

**Inference latency.** The inference latency directly determines the efficiency of the DNN models, which can be a major concern in some applications. For example, in autonomous driving, the DNN models (e.g., LSTM) are commonly

Table 2. Details of the hyperparameters used in our case study

| Hyperparameter | Hyperparameter search space | | | | Description |
|---|---|---|---|---|---|
| | CNN image classification | Resnet-50 | CNN text classification | LSTM sentiment classification | |
| **Architecture-related hyperparameters** | | | | | |
| Number of filters/units | 6, 12(default), 24, 48, 96, 192 | 16, 32, 64, 128(default), 256, 512, 1024 | 8, 16, 32, 64,128(default), 256 | 16, 32, 64(default),128, 256 | It is the main measure of DNN model's learning capacity. |
| Kernel size | 3, 5, 7, 9 (default 3 for CNN image classification and 7 for other models) | | | n/a | This determines the receptive field of a Convolutional Neural Network. |
| Embedding dimension | n/a | | 8, 16, 32, 64,128(default), 256 | 16, 32, 64,128(default), 256 | This defines the size of the vector space in which each word will be embedded. |
| **Layer-level model training decisions** | | | | | |
| Dropout ratio | [0.0, 0.9] (default 0.5) | | | | This hyperparameter is the fraction of the input units to drop, which helps prevent overfitting. |
| Activation function | tanh, relu, sigmoid (default tanh for LSTM sentiment classification and relu for other models) | | | | It is mathematical equation that determine the output of each neuron in neural network. |
| Pooling method | max(default), avg | max, avg, globalmax, globalavg(default) | globalmax(default), globalavg | n/a | This is required to down sample the detection of features in feature maps. |
| **Optimizer hyperparameters** | | | | | |
| Loss function | categorical crossentropy (default), poisson, kullback leibler divergence | | binary crossentropy (default), poisson, kullback leibler divergence | | This is to compute the quantity that a model should seek to minimize during training. |
| Optimizer | SGD, RMSprop, Adam (default), Adadelta, Adagrad, Adamax, Nadam, Ftrl | | | | It updates the weight parameters to minimize the loss function. |
| Learning rate | [1e-4, 1e-1] (default 1e-2 for Resnet-50 and1e-3 for other models) | | | | It determines the step size of adjusting the weights with respect to the loss gradient. |
| Batch size | 1, 2, 4, 8, 16, 32, 64, 128 (default), 256, 512 | 16, 32 (default), 64, 128, 256, 512 | | | This is the size of samples to process before internal model parameters are updated. |
| Number of epochs | [1, 100] (default 4) | [1, 100] (default 25) | [1, 100] (default 3) | [1, 100] (default 2) | It defines the number of times to work through the entire training dataset. |

used to predict the future trajectories to help the autonomous vehicle make appropriate navigation decisions [60]. If the DNN model cannot make predictions in a fast and efficient manner, such results may lead to unimaginable serious consequences. When measuring the inference latency of a DNN model, in order to eliminate the impact of environment noises, instead of recording the prediction latency for each individual testing sample once, we measure the inference latency of the entire testing dataset for 30 times and take the median value.

**Model size.** The size of a DNN model is an important factor when deploying the DNN model in production since in many cases the production resources are quite limited due to the production cost and environments. In particular, for mobile devices in which resources are rather constrained, the model size becomes a major concern and DNN models even need to be compressed before being deployed into the production environment. Thus, we measure the actual size of a DNN model occupied on the disk in megabytes.

**Floating point operations (FLOPs).** FLOPs is commonly-used to measure the DNN model complexity and efficiency [8, 54, 64] and it can also help practitioners gain better understanding about the energy aspect of DNN models [35, 47, 87]. FLOPs is easy to compute and can be done statically, which is independent of the underlying hardware and software configurations. Therefore, we measure this property by simply calculating how many computations a DNN model does.

**Battery consumption.** Energy and power are important when executing DNN models in mobile devices with the limited battery capacity [79]. In our study, in addition to the FLOPs, we further measure the battery consumption of the DNN models. Specifically, we implement a simple Android application that launches the DNN models to perform inference and then utilize the Android *dumpsys batterystats* tool to extract the battery usage (in milliampere-hour, mAh) of the Android application while the DNN models are performing inference. Before each run of our application, we first reset the battery statistics, then record the battery consumption after the execution completes.

### 3.5 Hardware and platforms

Our work focuses on studying the impact of tuning different hyperparameters on the performance of DNN models deployed across various hardware platforms. Thus, we first use the popular hyperparameter tuning toolkit called *Keras Tuner* [66] to automatically search for the optimal model hyperparameters and train the models on the servers that have strong computing power. Specifically, we observe that the model building time for training and tuning the studied hyperparameters once (e.g., tuning all hyperparameters) and generating 100 DNN models for *CNN image classification*, *Resnet-50*, *CNN text classification*, and *LSTM sentiment classification* are 1,380 minutes, 1,614 minutes, 326 minutes, and 10,080 minutes respectively. We then deploy the trained DNN models and further optimized models on multiple platforms including servers and mobile devices to evaluate the various performance aspects of these DNN models for inference.

**Server.** Our experiments of tuning hyperparameters, optimizing DNN models, and inference on servers are performed on a cluster consisting of two computing nodes, which both run the Scientific Linux release 7.8 (Nitrogen) operating system with NVIDIA CUDA 10.2 and cuDNN 7.6 installed. Each computing node is equipped with a 72-core 2.5 GHz Intel Xeon Gold 6248 CPU, 450 GB of RAM, and 8 NVIDIA Tesla V100 32G-GPU cards. The subject models are built with the Keras [23] API using TensorFlow [5] (version 2.2.0) as the back-end platform.

**Mobile.** The experiments on the mobile devices are conducted on two Xiaomi Mi 9 Android phones. These smartphones have the same software and hardware configurations, which include one Snapdragon 855 Octa-core Max 2.84 GHz processor, 8 GB RAM , 3300 mAh battery, and run the MIUI 12 operating system based on Android 10. In order to perform inference of DNN models on Android, we use the TensorFlow Optimization Toolkit to convert the standard

TensorFlow models to the TensorFlow Lite format before deploying the models on Android devices and utilize the latest version (current is 2.3.0) of the TensorFlow Lite library in our Android application as the back-end framework.

## 4 EMPIRICAL STUDY RESULTS

In this section, we present the empirical study results by answering two research questions (RQs).

### 4.1 RQ1: What is the impact of tuning different hyperparameters on the performance of DNN models?

*4.1.1 Motivation.* In order to search for the optimal DNN models to meet specific performance requirements in deep learning development, practitioners often systematically explore diverse combinations of configurations (i.e., hyperparameters) of the DNN models using automatic hyperparameter tuning tools (e.g., Keras Tuner). However, different usage scenarios of the DNN-based systems may need to satisfy different performance requirements, for example, in auto-driving vehicles, DNN models are required to make predictions in a fast and efficient manner, while in DNN-based authentication systems, inference accuracy would be a major concern. Besides, determining which hyperparameters to tune for specific performance requirements is still challenging, since tuning a DNN model with wrong types and ranges of hyperparameters may not achieve expected results, while putting too many types of hyperparameters and/or a large range of values into tuning would make the whole deep learning development process very expensive and may delay the entire release schedule of the software system, especially in a fast-paced release cycle [33]. Therefore, in our first RQ, we would like to study how the DNN models are affected by tuning different hyperparameters in terms of different aspects of performance.

*4.1.2 Approach.* To answer this RQ, we first apply hyperparameter tuning on our four subject DNN models, then we deploy these DNN models on the server platform and measure different performance properties of each DNN model when performing inference on the testing samples. Finally, we examine the performance differences between the models that tune each hyperparameter and the models that do not have this hyperparameter tuned, to understand the overall impact of that single hyperparameter on different DNN performance properties. The details of the subject DNN models and their corresponding used datasets are summarized in Table 1 of Section 3.2. Below, we describe each step of our evaluation approach for RQ1 in detail.

**Tuning DNN model hyperparameters.** For each DNN model, we perform hyperparameter tuning with several commonly-used hyperparameters in practice which cover different dimensions, including architecture-related hyperparameters, layer-level model training decisions, and optimizer hyperparameters (cf. Section 3.3). In particular, when experimenting with different combinations of hyperparameters, we resemble a more realistic situation where developers tune the hyperparameters of DNN models: they usually choose a set of hyperparameters from different dimensions to tune rather than tuning just one specific hyperparameter. Thus, in our work, we adopt a similar strategy of tuning DNN model hyperparameters to study the impact of tuning different hyperparameters on the DNN model in terms of different performance properties. Specifically, we first regard the performance properties of the DNN models resulting from tuning all the hyperparameters as the baseline, and then compare the focused performance properties of the tuned DNN models generated when fixing the hyperparameters in one dimension or fixing a single hyperparameter with the baseline. To avoid the bias of subjectively choosing the hyperparameter values when fixing a hyperparameter or a dimension of hyperparameters, we opt to use the hyperparameter values adopted in the official examples [2], which are the hyperparameter values chosen by the developers and experts of these models. If the official example does not provide a specific value for the hyperparameter, the default value in the DNN framework API code is used. In this case,

our work would provide insights for developers about how not tuning one specific hyperparameter or one dimension of hyperparameters impact the DNN model's different performance properties and help them understand the overall impact of each dimension and each single hyperparameter, and hence, would provide guidance for their hyperparameter tuning tasks and save them time and effort on these tasks. It is noteworthy that we respectively call the DNN models resulting from tuning all the hyperparameters, from fixing each dimension of hyperparameters and each hyperparameter at a time while tuning all other hyperparameters, as *tuning-all*, *fix-one-dimension*, and *fix-one-hyperparameter* in the rest of this paper. Notably, when tuning the hyperparameters of each DNN model, we apply the *random search* algorithm to randomly sample 100 trials of hyperparameter combinations and measure the corresponding model performance (cf. Section 3.3).

**Measuring DNN model performance.** After each hyperparameter tuning (i.e., *tuning-all*, *fix-one-dimension*, and *fix-one-hyperparameter*), we generate 100 DNN models with different combinations of hyperparameters. Subsequently, we deploy these models on the server and measure the performance properties of each DNN model, i.e., inference accuracy, inference latency, model size on the disk, and FLOPs. It is also worth noting that, to minimize the noise from the system warm-up and cool-down periods, for each DNN model, we repeat the model inference on the testing dataset 30 times and take the median value of all measured inference latency results as the final performance property of that model.

To gain an overall understanding of the impact of tuning different hyperparameters on the performance of a DNN model, we calculate and compare the distribution of different performance properties of the DNN model generated from tuning all studied hyperparameters. Due to the fact that different performance properties have different scales, we do not directly compare the distribution of these performance properties. Instead, we leverage the following approach to scale the performance properties[3]:

$$Px_{scaled} = \frac{Px}{Min(P)}$$

where $P$ is the performance property vector (i.e., measured after tuning all hyperparameters for each of the subject DNN models and it has the size of 100) that needs to be scaled, $Px$ is the $x^{th}$ value in the $P$ vector, and $Min(P)$ is the minimum value of the $P$ vector. It should be noted that we manually checked the measured performance properties in our experiments, and all the values are above zero, thus, there is no problem of division by zero. We choose this scaling approach as compared to other scaling approaches (e.g., min-max scaling [73]), it not only preserves the range and the linear relationship in the original data, but also provides better readability for visualization. For example, the inference accuracy of best-10 DNN models may be very similar, the min-max scaling would group them together, leading to poor readability, whereas our scaling approach can provide a better visualization for these small differences in the performance properties. In addition to measuring the performance distribution of all generated DNN models after tuning all hyperparameters (i.e., 100 models), we also select and compare the different performance properties of the best 10 models (based on the training time accuracy), since these models achieve the best performance on the objective metrics (e.g., accuracy or mean squared error) and are more likely to be selected as the final model than the rest of the models.

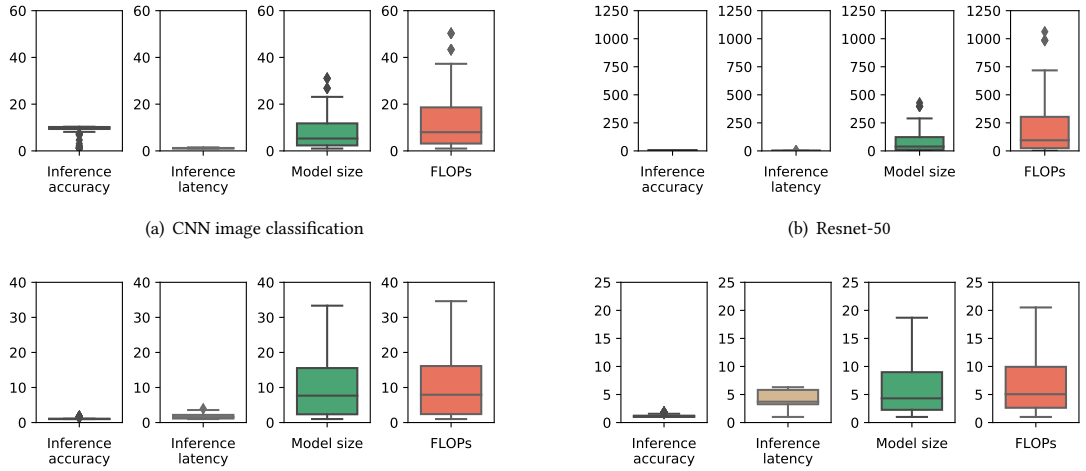
**Comparing the performance properties of the *fix-one-dimension* and *fix-one-hyperparameter* models with the *tuning-all* models.** In order to further understand the impact of each dimension of hyperparameters and each hyperparameter on the DNN model performance in terms of different properties, we set the generated DNN

[3]Please note that this scaling approach only applies to the comparison between different DNN performance properties (i.e., Figure 2, Figure 3, and Figure 6).

models from tuning all the hyperparameters (i.e., the *tuning-all* models) as the baseline, and compare the performance of the DNN models resulted from tuning all but one dimension of hyperparameters (i.e., the *fix-one-dimension* models) and tuning all but one hyperparameter (i.e., the *fix-one-hyperparameter* models) with the baseline. We select the best 10 models from the resulting models (i.e., 100 models) after each hyperparameter tuning, since they often achieve similar performance on objective metrics (e.g., accuracy or loss) and are more likely to be selected as the final model than the rest of the models. Our intuition is that: if the *fix-one-dimension* or *fix-one-hyperparameter* DNN models have significantly different performance on specific properties compared to the *tuning-all* models, then the fixed dimension or hyperparameter has a significant impact on the performance properties. We first calculate the best model relative difference from the two different DNN model groups. It is calculated by the performance property of the best model from the group of DNN models that are obtained from *fix-one-dimension* or *fix-one-hyperparameter* minus the best model's performance property from the group that contains the DNN models generated by *tuning-all*, normalized by the latter. Such a relative difference can be interpreted as how not tuning one specific hyperparameter or one specific dimension of hyperparameters impacts the resulting best model of the hyperparameter tuning process. For example, in terms of inference accuracy, if this value is negative, it means if we choose not to tune this specific hyperparameter or specific dimension of hyperparameters, the target performance property of the best model will not be as good as when we tune it/them. While regarding other studied performance properties (i.e., inference latency, model size, FLOPs, and battery consumption), if this value is negative, it means that not tuning the target hyperparameter or target dimension of hyperparameters would achieve a better target performance than tuning it/them.

In order to examine whether such performance differences between the *fix-one-dimension* or *fix-one-hyperparameter* models and the *tuning-all* DNN models are introduced by chance, and to understand the scale of these differences, we compare the two performance property distributions of the best-10 DNN models from each group using statistical tests and effect sizes, similar to previous studies [20, 55]

- **Statistical test.** In order to evaluate the impact of tuning one hyperparameter or one dimension of hyperparameters on DNN model performance, we use the Mann-Whitney U test [65] as it is non-parametric and it does not assume a normal distribution of the compared data, to determine whether there exists a statistically significant difference (i.e., p-value $< 0.05$) between the performance properties (e.g., inference accuracy or inference latency) from models generated from fixing one hyperparameter or fixing one dimension of hyperparameters and the ones when tuning all studied hyperparameters. In particular, each comparison group consists of 10 values for the corresponding best 10 models, and for inference latency, each value is the median from the 30 times of repetition. It is worth noting that, in practice, Bonferroni correction [16] is usually used together with the statistical test (i.e., Mann-Whitney U test) to counteract the problem of multiple comparisons: while a given p-value may be appropriate for each individual comparison, it is not for the set of all comparisons. In this correction approach, the p-value threshold needs to be lowered to account for the number of comparisons being performed and thus, to avoid spurious positives. However, our approach of studying the impact of tuning different hyperparameters on the performance properties of DNN models only involves comparison of the performance properties from two different groups once at a time, i.e., DNN models generated from *fix-one-dimension* or *fix-one-hyperparameter* (e.g., fixing learning rate) and the ones built from *tuning-all*. Therefore, our study does not apply multiple comparisons where several dependent or independent statistical tests are being performed simultaneously and correspondingly, Bonferroni correction is not applicable in our empirical case study.




(a) CNN image classification

(b) Resnet-50

(c) CNN text classification

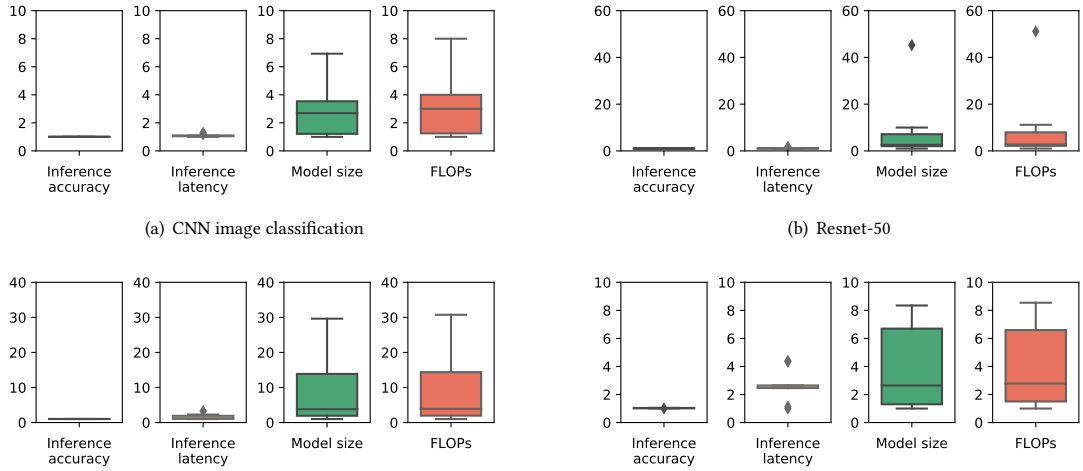
(d) LSTM sentiment classification

Fig. 2. (Scaled) performance properties distribution of standard 100 DNN models

Note: For each subject model, all the performance properties are normalized by the minimum value of the corresponding property in order to visually compare the distributions among each other.

- **Effect size.** Reporting only the statistical significance may lead to misleading results, i.e., if the sample size is very large, the p-value can be very small even if the difference is trivial. Therefore, we apply Cliff's Delta [25] to quantify the effect size of the difference between the specific performance properties of the models generated from fixing one hyperparameter or fixing one dimension of hyperparameters and the ones when tuning all studied hyperparameters.

*4.1.3 Results.* **Hyperparameter tuning has a significant influence on the performance of the studied DNN models.** Figure 2 shows the distributions of the scaled performance properties (i.e., inference accuracy, inference latency, model size, and FLOPs) of our subject DNN models (i.e., *CNN image classification*, *Resnet-50*, *CNN text classification*, and *LSTM sentiment classification*) after tuning all the studied hyperparameters. We find that the performance of generated DNN models is spread over a considerably wide range in terms of different aspects, which demonstrates the significant impact of hyperparameter tuning on the DNN model performance. In particular, as shown in Figure 2(b), tuning hyperparameters on the *Resnet-50* model achieves the largest differences in the scaled performance property values between the first and third quartiles in both model size and FLOPs (i.e., a difference of 112.62 in model size and a difference of 279.78 in FLOPs); while for inference accuracy and inference latency, *CNN image classification* and *LSTM sentiment classification* respectively reach the largest differences in the scaled performance property values between the first and third quartile, with the difference values as much as 2.56 (i.e., for the *LSTM sentiment classification* model). Our work confirms prior work's finding that hyperparameter tuning has a significant impact on the accuracy property of the DNN models [86]. As the tuned hyperparameters (e.g., architecture-related hyperparameters) directly control the structure and complexity of the tuned DNN models, they also have significant impacts on the other properties (e.g., inference latency) of the tuned DNN models.




(a) CNN image classification

(b) Resnet-50

(c) CNN text classification

(d) LSTM sentiment classification

Fig. 3. (Scaled) performance properties distribution of standard best-10 DNN models

Note: For each subject model, all the performance properties are normalized by the minimum value of the corresponding property in order to visually compare the distributions among each other.

**The top DNN models resulted from tuning hyperparameters have very similar accuracy, while other performance properties can differ significantly.** For each of our four subject models, we present the scaled performance properties distributions of the best 10 DNN models resulted from tuning all the studied hyperparameters in terms of training time accuracy in Figure 3. The results show that, for all our subject models, while there are only small variances in inference accuracy among these top DNN models, the other performance properties (e.g., inference latency or model size) may spread over a comparatively wide range. Specifically, as shown in Figure 5(c), the *CNN text classification model* achieves a distribution of inference accuracy with a first quartile of 1.01 and a third quartile of 1.02, while it achieves a distribution of inference latency with a first quartile of 1.06 and a significantly higher third quartile of 1.93. Such a finding can by explained by the reason that existing DNN hyperparameter tuning tools (e.g., Keras Tuner) usually only support accuracy-oriented search objectives (e.g., accuracy or loss). Thus, the highly ranked models (i.e., ranked by the search objective) from the tuning results tend to provide similar accuracy. However, the subtle difference in accuracy may hide significantly larger differences in other performance properties. Therefore, based on this finding, we would suggest that although accuracy is a vital factor when constructing DNN models, practitioners should not always choose the DNN model with the best accuracy, as other DNN models (with different hyperparameter configurations) showing slightly weaker accuracy may have significantly better improvements in terms of other performance properties (e.g., faster inference speed).

**The impact of tuning different combinations of hyperparameters on different DNN model performance properties.** Table 3 shows the detailed results of comparing the performance distributions of our studied DNN models obtained from two groups: the first group contains the DNN models generated by tuning all hyperparameters and the second group of DNN models is obtained from fixing one hyperparameter or one dimension of hyperparameters while tuning all other hyperparameters. For each model, we present the results on four performance properties, i.e., inference accuracy, inference latency, model size, and FLOPs. The sub-columns "Best model relative difference" show

the relative difference of the best models from the two different groups and the "p-value/Effect size" sub-columns show the statistical significance or the effect size of the difference between the performance properties of all the DNN models from those two groups. It is noted that if there exists a statistically significant difference (i.e., p-value < 0.05), we present the magnitude of the difference (i.e., effect size), otherwise we show the p-value.

**1) The same hyperparameter or same dimension of hyperparameters can cause different impact on the performance of different DNN models.** As shown in Table 3, we observe that even for the same performance property of the different DNN models, the resulting model performance can be affected differently by tuning the same hyperparameters or the same dimension of hyperparameters. For example, in terms of inference accuracy, there exist hyperparameters in all three dimensions that introduce the significant impact (i.e., large effect size) for *CNN image classification*, *Resnet-50*, and *CNN text classification*. However, the inference accuracy of the *LSTM sentiment classification* model is only influenced by the optimizer hyperparameters. Moreover, in terms of inference latency, for *CNN image classification* and *Resnet-50*, all three dimensions of hyperparameters lead to significant impact, while for *CNN text classification* and *LSTM sentiment classification*, only optimizer and layer-level hyperparameter cause notable effect, respectively. Although the impact may not be significant when fixing all hyperparameters in the target dimension, it does have a significant influence when fixing specific hyperparameters in that dimension. The reason behind this finding would be that as different DNN models have different structures, they may have different sensitivity to the same hyperparameters. For example, a more complex model may be more sensitive to the dropout ratio which controls the overfitting of the models, thus the dropout ratio hyperparameter has a larger impact on the CNN image classification (large effect size for inference latency) and Resnet-50 (large effect size for inference accuracy and latency) models. Hence, such a finding would imply that the impression of the impact of certain hyperparameters on DNN model performance learned from one model may not be directly applicable to a different model.

**2) The same hyperparameter or same dimension of hyperparameters may have different impact on different performance properties of the same DNN model.** For instance, as shown in Table 3, architecture-related hyperparameters lead to remarkable impact on the inference accuracy, model size, and FLOPs of the *CNN text classification* model, yet not on the inference latency. For the *Resnet-50* model, the hyperparameter "Pooling method" causes significant impact on all other performance properties except for the inference accuracy. Such a finding would be explained by the reason that the different performance properties of a DNN model can be mutually conflicting (e.g., a complex model may have a better accuracy but with bigger size and longer inference latency), thus they can be impacted by the same hyperparameters in different ways, for example, compared to learning rate, number of filters/units that control the structure of the network is more likely to have a significant impact on model size and FLOPs than inference accuracy. Therefore, based on this finding, we would suggest that practitioners should not just consider one DNN model performance property when choosing the hyperparameters to tune.

**3) Tuning the same hyperparameter or same dimension of hyperparameters lead to the similar impact on model size and FLOPs.** As illustrated in Table 3, we can clearly see that, for all our four subject DNN models, the model size and FLOPs are almost equally influenced by the architecture-related hyperparameters, for instance, fixing the value of "Number of filters/units" for *LSTM sentiment classification* model leads to similar impact between model size and FLOPs in both best model relative difference (i.e., 0.015 and 0.018 for model size and FLOPs respectively) and effect size (i.e., both are large). In addition, for *Resnet-50*, those two performance properties (i.e., model size and FLOPs) are also significantly impacted by layer-level model training decisions (i.e., pooling method) and optimizer hyperparameters (i.e., loss function and the number of epochs). By further investigating the reason behind such a result, we find that a DNN model is essentially a collection of mathematical functions that are structured by deep learning frameworks

(e.g., TensorFlow or PyTorch) as tensor-oriented computation graphs. A larger model size usually indicates that the network has a relatively more complex structure and/or more compute nodes in the graphs, and thus requires more computations in the DNN model, i.e., higher FLOPs. Therefore, to some extent, there is a relationship between the model size and FLOPs, and tuning the same hyperparameter or same dimension of hyperparameters would have a similar impact on these two performance properties. Such a result also agrees with a recent study [14] that performs benchmark analysis of some existing DNN models proposed in the state-of-the-art for image recognition.

Summary of RQ1

While tuning the hyperparameters has a significant impact on the performance of all the studied DNN models, the impact of tuning specific hyperparameters varies across different DNN models. In addition, we observe that, although the top tuned DNN models have very similar accuracy, they may have significantly different performance in terms of other aspects (e.g., inference latency). Thus, practitioners should not always choose the best tuned model, but instead consider other performance properties while choosing the most appropriate model from the tuning results.

### 4.2 RQ2: What is the combined impact of hyperparameter tuning and model optimization on the performance of optimized DNN models?

*4.2.1 Motivation.* RQ1 shows that tuning different hyperparameters of DNN models can impact the model performance in terms of different aspects. However, the standard DNN models generated by hyperparameter tuning process may not be suitable for direct deployment in production due to various environment restrictions, e.g., processing, memory, power consumption, or model storage space. Instead, they may need to be optimized for deployment and execution in a particular production environment, especially on resource-bounded mobile devices. However, tuning the hyperparameters may have different impact on the standard and the optimized models in terms of various performance properties. Thus, the goal of this RQ is to assess the combined influence of tuning different hyperparameters and model optimization on the performance of optimized DNN models for mobile devices.

*4.2.2 Approach.* In order to understand how the performance of optimized DNN models is affected by tuning different hyperparameters, we perform pruning, quantization, and encoding on the models resulted from different hyperparameter tuning trials (i.e., *tuning-all*, *fix-one-dimension*, and *fix-one-hyperparameter*). Then, we compare the focused performance properties of the optimized models that are resulted from these different hyperparameter tuning trials.

**Optimizing standard DNN model.** In order to better study the impact of tuning different hyperparameters on models with different DNN model optimization techniques applied, for each of the subject DNN models, we carry out the optimization process in three steps, including pruning, quantization, and encoding, then obtain the optimized models (with all three model optimization techniques applied together) that are suitable for deployment and execution on mobile devices. In particular, we first prune the standard model generated from hyperparameter tuning by using the TensorFlow Model Optimization Toolkit, in which the pruning is performed to the whole model (i.e., all layers in the model). We opt to prune the DNN models with 80% sparsity (i.e., 80% zeros in weights) to get the optimal result. We would like to note that our choice of sparsity (i.e., 80% sparsity) is inspired by the official tutorial about pruning in TensorFlow Keras [3]. From the result of the tutorial, we observe that by applying pruning on DNN models with such

Table 3. Overall impact of tuning hyperparameters on the performance of standard best-10 DNN models

(a) Result for CNN image classification

| Hyperparameter | Inference accuracy | | Inference latency | | Model size | | FLOPs | |
|---|---|---|---|---|---|---|---|---|
| | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size |
| **Architecture-related hyperparameters** | -0.008 | large | -0.031 | 0.455 | -0.184 | large | -0.214 | large |
| Number of filters/units | -0.001 | 0.325 | -0.083 | large | -0.422 | large | -0.500 | large |
| Kernel size | -0.003 | large | 0.159 | 0.455 | 4.480 | 0.258 | 5.287 | 0.258 |
| **Layer-level model training decisions** | 0.002 | large | -0.099 | 0.260 | 0.847 | 0.096 | 1.000 | 0.089 |
| Dropout ratio | 0.001 | 0.056 | -0.086 | large | 1.128 | 0.365 | 1.329 | 0.350 |
| Activation function | 0.002 | large | 0.188 | 0.396 | 5.934 | 0.191 | 7.000 | 0.221 |
| Pooling method | 0.001 | 0.500 | -0.088 | 0.485 | 2.542 | 0.380 | 3.000 | 0.337 |
| **Optimizer hyperparameters** | -0.004 | large | 0.084 | large | 2.542 | 0.134 | 3.000 | 0.133 |
| Loss function | $\ll 0.001$ | 0.106 | -0.037 | 0.455 | 0.847 | 0.439 | 1.000 | 0.485 |
| Optimizer | $\ll 0.001$ | 0.106 | -0.082 | 0.396 | 0.001 | 0.455 | $\ll 0.001$ | 0.469 |
| Learning rate | -0.001 | 0.440 | 0.179 | 0.425 | 7.050 | 0.440 | 8.317 | 0.455 |
| Batch size | 0.001 | 0.248 | -0.024 | 0.312 | 2.544 | 0.213 | 3.000 | 0.270 |
| Number of epochs | -0.004 | large | 0.219 | 0.396 | 8.340 | 0.172 | 9.841 | 0.223 |

(b) Results for Resnet-50

| Hyperparameter | Inference accuracy | | Inference latency | | Model size | | FLOPs | |
|---|---|---|---|---|---|---|---|---|
| | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size |
| **Architecture-related hyperparameters** | 0.001 | 0.145 | 0.302 | large | 2.845 | large | 2.912 | large |
| Number of filters/units | 0.030 | 0.260 | 0.116 | large | 2.845 | large | 2.911 | large |
| Kernel size | -0.026 | large | -0.463 | large | -0.805 | 0.312 | -0.822 | 0.312 |
| **Layer-level model training decisions** | 0.002 | 0.485 | -0.142 | 0.052 | -0.182 | 0.396 | -0.186 | 0.396 |
| Dropout ratio | 0.006 | large | -0.078 | large | 0.501 | 0.172 | 0.511 | 0.172 |
| Activation function | 0.057 | 0.339 | 0.023 | large | 0.621 | 0.192 | 0.634 | 0.192 |
| Pooling method | 0.040 | 0.485 | 0.294 | large | 2.588 | large | 2.649 | large |
| **Optimizer hyperparameters** | 0.011 | 0.485 | 0.661 | 0.172 | 7.486 | 0.052 | 7.671 | 0.052 |
| Loss function | 0.002 | 0.154 | 0.399 | large | 3.422 | large | 3.422 | large |
| Optimizer | -0.012 | 0.455 | -0.067 | 0.396 | -0.625 | 0.093 | -0.639 | 0.093 |
| Learning rate | 0.045 | 0.070 | 0.120 | 0.093 | 0.628 | 0.106 | 0.642 | 0.106 |
| Batch size | 0.003 | 0.367 | 0.197 | 0.396 | 1.431 | 0.061 | 1.465 | 0.061 |
| Number of epochs | 0.014 | large | 0.540 | large | 4.891 | large | 5.007 | large |

Note: for column "p-value/Effect size", if p-value smaller than 0.05, we present the effect size, else we show the p-value.

Table 3. Overall impact of tuning hyperparameters on the performance of standard best-10 DNN models (cont.)

(c) Results for CNN text classification

| Hyperparameter | Inference accuracy | | Inference latency | | Model size | | FLOPs | |
|---|---|---|---|---|---|---|---|---|
| | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size |
| **Architecture-related hyperparameters** | $\ll 0.001$ | 0.214 | -0.485 | 0.061 | -0.453 | large | -0.454 | large |
| Number of filters/units | 0.005 | 0.485 | -0.674 | 0.485 | -0.907 | 0.285 | -0.909 | 0.285 |
| Kernel size | $\ll 0.001$ | large | -0.050 | 0.425 | 0.027 | 0.172 | 0.027 | 0.172 |
| Embedding dimension | 0.005 | 0.172 | -0.416 | 0.061 | -0.497 | large | -0.497 | large |
| **Layer-level model training decisions** | 0.002 | 0.339 | -0.572 | 0.106 | -0.747 | 0.485 | -0.748 | 0.485 |
| Dropout ratio | 0.003 | 0.061 | -0.699 | 0.106 | -0.923 | 0.485 | -0.924 | 0.485 |
| Activation function | 0.011 | 0.154 | -0.604 | 0.154 | -0.874 | 0.070 | -0.875 | 0.070 |
| Pooling method | 0.011 | large | -0.554 | 0.425 | -0.686 | 0.260 | -0.687 | 0.260 |
| **Optimizer hyperparameters** | 0.023 | large | -0.672 | 0.367 | -0.866 | 0.339 | -0.867 | 0.312 |
| Loss function | 0.001 | 0.455 | -0.746 | 0.121 | -0.960 | 0.285 | -0.961 | 0.285 |
| Optimizer | $\ll 0.001$ | 0.485 | -0.645 | large | -0.716 | 0.260 | -0.717 | 0.260 |
| Learning rate | -0.019 | 0.236 | -0.710 | 0.061 | -0.928 | 0.425 | -0.929 | 0.425 |
| Batch size | -0.004 | 0.136 | -0.697 | large | -0.921 | 0.312 | -0.923 | 0.312 |
| Number of epochs | -0.013 | large | -0.515 | 0.260 | -0.638 | 0.093 | -0.639 | 0.093 |

(d) Results for LSTM sentiment classification

| Hyperparameter | Inference accuracy | | Inference latency | | Model size | | FLOPs | |
|---|---|---|---|---|---|---|---|---|
| | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size |
| **Architecture-related hyperparameters** | 0.006 | 0.325 | -0.044 | 0.137 | -0.487 | 0.221 | -0.483 | 0.221 |
| Number of filters/units | -0.001 | 0.202 | -0.593 | 0.214 | 0.015 | large | 0.018 | large |
| Embedding dimension | -0.011 | 0.410 | 0.622 | 0.285 | -0.430 | 0.163 | -0.379 | 0.163 |
| **Layer-level model training decisions** | -0.017 | 0.236 | -0.527 | large | 0.102 | 0.236 | 0.149 | 0.260 |
| Dropout ratio | -0.015 | 0.410 | -0.647 | 0.236 | -0.868 | 0.367 | -0.865 | 0.367 |
| Activation function | -0.009 | 0.353 | -0.544 | large | -0.802 | 0.425 | -0.750 | 0.367 |
| **Optimizer hyperparameters** | -0.002 | large | 0.647 | 0.312 | 0.040 | 0.425 | 0.053 | 0.425 |
| Loss function | -0.002 | 0.213 | -0.076 | 0.485 | -0.914 | 0.260 | -0.899 | 0.260 |
| Optimizer | -0.025 | 0.440 | -0.599 | 0.396 | -0.483 | 0.113 | -0.472 | 0.113 |
| Learning rate | 0.007 | 0.247 | -0.040 | 0.285 | -0.423 | 0.203 | -0.375 | 0.163 |
| Batch size | -0.002 | 0.075 | -0.663 | 0.455 | -0.872 | 0.121 | -0.872 | 0.137 |
| Number of epochs | -0.014 | 0.182 | 0.602 | 0.137 | -0.874 | 0.312 | -0.875 | 0.312 |

Note: for column "p-value/Effect size", if p-value smaller than 0.05, we present the effect size, else we show the p-value.

configuration, one can create a significantly smaller and sparser DNN model, while with minimal accuracy difference. However, our experiments and results have no restriction on the pruning configurations and practitioners can apply their preferred choice of sparsity to the pruning process in the DNN optimization pipeline. Then, we convert the precision of weights from 32-bit floats to 8-bit integers in post-training model quantization, which is the commonly-used approach in practice. By default, our used deep learning back-end framework, i.e., TensorFlow, supports such quantization operations, except for the LSTM layer, therefore, we skip quantization step on the pruned LSTM models. Afterwards, we use the TensorFlow Optimization Toolkit to convert the standard TensorFlow models to the TensorFlow Lite format before deploying the models on mobile devices. Finally, we apply Huffman and LZ77 encoding to further compress the DNN models, i.e., reduce the model size.

**Measuring the performance of optimized DNN models.** After applying DNN model optimization techniques, for each standard DNN model, we generate the corresponding optimized model with all three optimization methods (i.e., pruning, quantization, and encoding) applied in order to fully study the effects of tuning different hyperparameters on the DNN models with different aspects of optimizations. Therefore, we first deploy the optimized DNN models on the mobile devices, and then measure the performance properties of each optimized DNN model when it is under prediction workload. In particular, for model size, we calculate it for both unencoded and encoded models. Similar to RQ1, we perform inference on the testing dataset 30 times and keep the median inference latency result to reduce the noise caused by system warm-up and cool-down.

After completing the measurement of the performance properties for all optimized DNN models, we follow the same process of calculating and comparing the different performance properties' distributions as in RQ1 (cf. Section 4.1.2) for each of our subject DNN models, to understand the impact of tuning different hyperparameters on the performance of the optimized DNN models.

**Comparing the performance property distributions between standard and optimized DNN models.** To evaluate whether the optimized DNN models perform differently from the corresponding standard models, we compare the distributions of different DNN models' performance properties in the standard form and optimized form. Our intuition is that if the set of DNN models have different distributions in performance properties before and after optimization, this may be an indicator that model optimization will cause changes in the performance characteristics of the DNN model. For example, if the standard DNN models have similar inference speed on the server side, after applying multiple optimization techniques on these models, they may have significantly different inference speed on mobile devices. In particular, we select the best 10 standard models from tuning all selected hyperparameters and the corresponding optimized models. Then, for each of the performance properties, we calculate the largest absolute relative deviation (LARD) of these two distributions as in the following:

$$LARD_P = \left| \frac{Best(P) - Worst(P)}{Worst(P)} \right|$$

where $P$ is the performance property vector (i.e., measured from the best 10 DNN models generated by tuning all hyperparameters for each of the subject DNN models or their corresponding optimized ones and it has the size of 10), Best(P) and Worst(P) are the best value and worst value of the $P$ vector, respectively. Specifically, for the inference latency, model size, and battery consumption of a DNN model, the lower the values are, the better performance they indicate; while for inference accuracy, the higher values mean better performance. It should be noted that the purpose of the LARD metric is not to show the performance difference between a standard DNN model and its optimized counterpart. Rather, we compare the LARD metric between the standard models and the optimized ones to understand

whether the optimization leads to larger differences among the performance of the models. We choose the LARD metric to measure the performance characteristics (i.e., the variance of the distributions) of DNN models as classical metrics (i.e., standard deviation or mean absolute deviation) that measure the variability or dispersion of a set of values are not suitable for the comparison between two performance datasets obtained from different platforms (i.e., standard DNN model on servers and optimized DNN model on mobile devices) with different scales, while LARD can preserve the range of a set of performance properties while reducing such bias.

In addition, to understand the distribution of the changes between standard DNN models and optimized DNN models with respect to various performance properties, for each of the DNN models, we measure the relative difference of performance properties (e.g., inference accuracy, inference latency) between the standard DNN model and the corresponding optimized one. It is calculated by the performance property of the optimized DNN model minus the performance property of the corresponding standard model, normalized by the latter. Besides, we also visualize the distribution of these relative performance property changes between standard and optimized DNN models using box plots.

In order to have a more comprehensive understanding of the difference between the distributions of different DNN models' performance properties in the standard form and optimized form, we perform further investigations and comparisons on the distributions of pairwise performance property differences between optimized models and standard models. In particular, for a pair of standard DNN models, $m_1$ and $m_2$, and the corresponding optimized ones, $m_1'$ and $m_2'$, we compute the relative difference for each of the performance properties (e.g., inference latency) between $m_1$ and $m_2$, normalized by $m_2$, and the corresponding difference between $m_1'$ and $m_2'$, normalized by $m_2'$. Then, we compare these two differences. Such a comparison can be expressed in the following formula:

$$\Delta_{pairwise} = \frac{m_1' - m_2'}{m_2'} - \frac{m_1 - m_2}{m_2}$$

We then repeat this process for all the combinations of the best-10 DNN models and the corresponding optimized ones, which leads to a total of 45 model pairs (i.e., ${}_{10}C_2$, which means the number of combinations when choosing two objects from the set of 10 objects). We also visualize the distributions of such differences using a density plot. In addition, we utilize the Kolmogorov-Smirnov test [77] to determine if there exists a statistically significant difference (i.e., p-value $<$ 0.05) between the distributions of the pairwise model performance property differences of the standard DNN models and optimized DNN models. We choose the Kolmogorov-Smirnov test since it does not enforce any assumptions on the distributions of the data.

**Comparing the performance properties of the optimized *fix-one-dimension* and *fix-one-hyperparameter* models with the optimized *tuning-all* models.** Similar to RQ1, we first set the performance of the optimized DNN models from tuning all the hyperparameters as the baseline, and then apply the same statistical analysis, including calculating best model relative difference, statistical testing and measuring effect size, to compare each focused performance property of the optimized DNN models generated when fixing the hyperparameters in one dimension or fixing one single hyperparameter at a time with the baseline, to study the impact of tuning different hyperparameters on the optimized DNN model in terms of different performance properties.

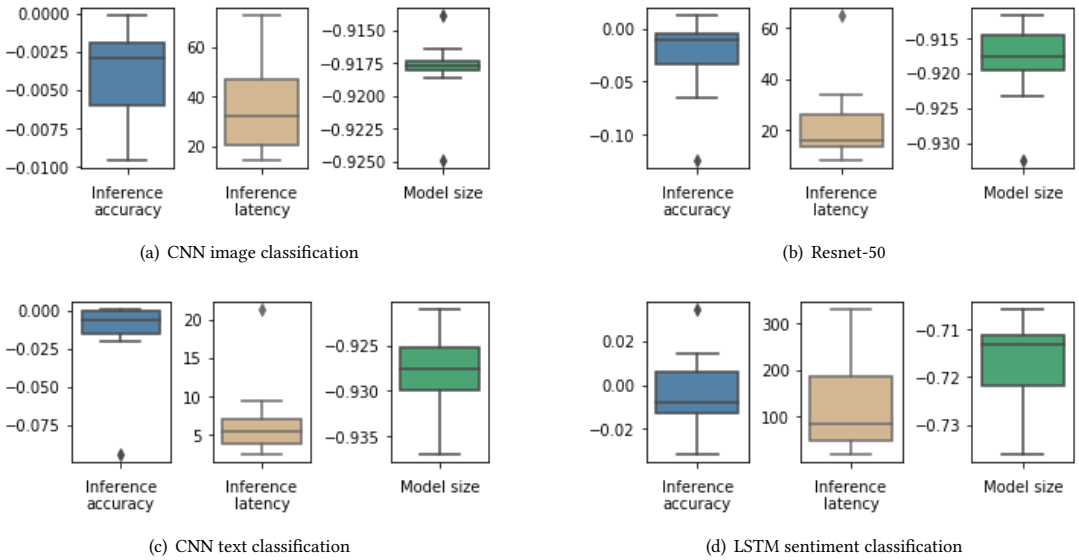
*4.2.3 Results.* **DNN model optimization can lead to significantly different performance distributions between the standard DNN models and the optimized ones.** Table 4 summarizes the comparison results of performance properties between the standard and the optimized DNN models for our four subject models. We find that after applying optimization techniques on the standard DNN models, the LARD for inference accuracy would be slightly

different with the deviation ranging from 0.02 to 0.11, except for the *CNN image classification* model which achieves the same LARD between the standard and optimized models. After further investigation of the distributions of inference accuracy, we suspect the reason being that the accuracy distributions are considerably concentrated, i.e., with the standard deviation of $\ll$0.001, in both the distributions of the standard and optimized models. For inference latency, we observe that there are notable differences in the distributions between the standard and the optimized DNN models. In particular, the *Resnet-50* model achieves the largest deviation in terms of LARD, with the value of 0.74 and 10.53 for standard models and optimized models, respectively. Such noteworthy difference can be interpreted as that even a set of DNN models have similar inference speed on the server or cloud platforms, they may have significant variance in inference latency on mobile devices. We also notice mild differences in the distributions of model sizes between the standard and the optimized DNN models. Particularly, the standard and optimized *LSTM sentiment classification* DNN models have quite close model size distributions, with a slight difference of LARD at only 0.24 (i.e., in terms of model size, the relative difference between the best and worst performance of the standard models are similar to the relative difference between the best and worst performance of the optimized models). Such results can be interpreted by the fact that we only apply standard conversion (transform a TensorFlow model to TensorFlow Lite model) on the pruned LSTM models to make them compatible for deployment on mobile devices, but without quantizing these LSTM models (cf. Section 4.2.2).

Figure 4 shows the distributions of the performance property changes between the standard DNN models and the optimized DNN models. We observe that the DNN model optimization introduces a significant impact on both inference latency and model size, while with relatively minor influence on inference accuracy. In particular, all of our four subject DNN models present remarkable differences in inference latency between the standard DNN models and the optimized ones, with a median relative difference[4] from 5.47 times (i.e., 547.03%) (in *CNN test classification*) to 83.54 times (i.e., 8,353.57%) (in *LSTM sentiment classification*). Regarding the model size, DNN optimization has a strong ability to reduce the standard model size. For example, considering the relative changes of the model size in *Resnet-50* models, the DNN model can be compressed by 91.16% or more compared to the standard model size. However, in terms of inference accuracy, although the DNN model optimization would bring some variation (often degradation) in the inference accuracy (i.e., the median value of the relative changes in inference latency for four subject DNN models are below 0), such variance is relatively small. Specifically, the median relative differences are -0.29%, -1.17%, -0.63%, and -1.16% for *CNN image classification*, *Resnet-50*, *CNN text classification*, and *LSTM sentiment classification*, respectively.

Figure 5 summarizes the distributions of the pairwise model performance property differences of the standard DNN models and optimized DNN models. Similar to Figure 4, from Figure 5 we observe that, the standard and optimized DNN models show relatively similar distributions of inference accuracy differences (i.e., the pairwise difference between the relative differences of two standard models and two optimized models (i.e., $\Delta_{pairwise}$) has distributions with most values gathering around 0). Yet, we still observe more than 10% (25% for the Resnet-50 model) differences between 1) the relative difference of two standard models and 2) the relative difference between the optimized counterparts of the two models. Furthermore, the distributions of other properties (i.e., inference latency and model size) can be more different between standard models and optimized ones (i.e., the pairwise difference distributions (i.e., $\Delta_{pairwise}$) have longer tails, reaching up to 1,000% difference for the inference latency of the Resnet-50 model). In addition, from the statistical test result (shown in Table 5), we find that all of our four subject DNN models (i.e., *CNN image classification*, *Resnet-50*, *CNN text classification*, and *LSTM sentiment classification*) have statistically significant differences (i.e., p-value

[4]Median relative difference is calculated as the median value of the differences of performance properties between each pair of optimized models and standard models normalized by standard models.




(a) CNN image classification

(b) Resnet-50

(c) CNN text classification

(d) LSTM sentiment classification

Fig. 4. Performance property relative difference distributions between optimized models and standard models normalized by standard models (best-10 models)

< 0.05) between the distributions of the pairwise model performance property differences of the standard DNN models and optimized DNN models. Such a result further supports our finding that DNN model optimization would lead to significantly different performance distributions between the standard DNN models and the optimized ones.

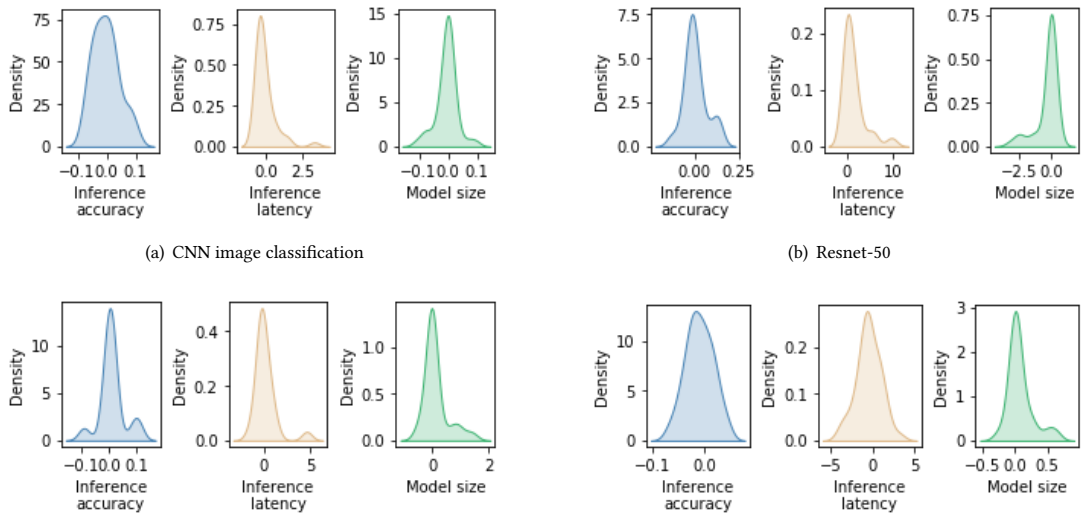
These findings can be explained by the reason that different DNN models, even different layers, would have different sensitivity to the optimization process. For example, since the number of parameters of convolution layers is intuitively less than the fully connected layers, convolution layers are more likely to be sensitive to pruning [38, 54]. Therefore, in terms of two DNN models with different combinations of hyperparameters, when they undergo the same model optimization configuration, there may be different degrees of optimization (especially pruning) on them, resulting in different distributions of the performance properties before and after optimization. Thus, based on these findings, we would suggest that practitioners need to pay attention to the significant performance differences between the optimized DNN models derived from the standard models with very similar performance.

Table 4. Results of comparing the LARD metric between standard DNN models (on servers) and optimized DNN models (on mobile devices) performance properties. A larger LARD metric indicates a larger variation of the performance properties.

| Property | CNN image classification | | Resnet-50 | | CNN text classification | | LSTM sentiment classification | |
|---|---|---|---|---|---|---|---|---|
| | Standard | Optimized | Standard | Optimized | Standard | Optimized | Standard | Optimized |
| Inference accuracy | 0.01 | 0.01 | 0.03 | 0.14 | 0.02 | 0.11 | 0.05 | 0.03 |
| Inference latency | 0.30 | 5.01 | 0.74 | 10.53 | 2.28 | 8.62 | 3.40 | 7.74 |
| Model size | 5.93 | 6.61 | 44.26 | 41.62 | 28.63 | 30.09 | 7.35 | 7.11 |

Note: Since there is no readily available tool support to calculate the FLOPs of optimized DNN models on mobile devices, we do not report the results of comparing the LARD metric in the FLOPs between standard DNN models and optimized DNN models.

**Although the accuracy of the top DNN models after optimization is very similar, other performance properties still differ significantly.** For each subject model, the distributions of various scaled performance properties




(a) CNN image classification

(b) Resnet-50

(c) CNN text classification

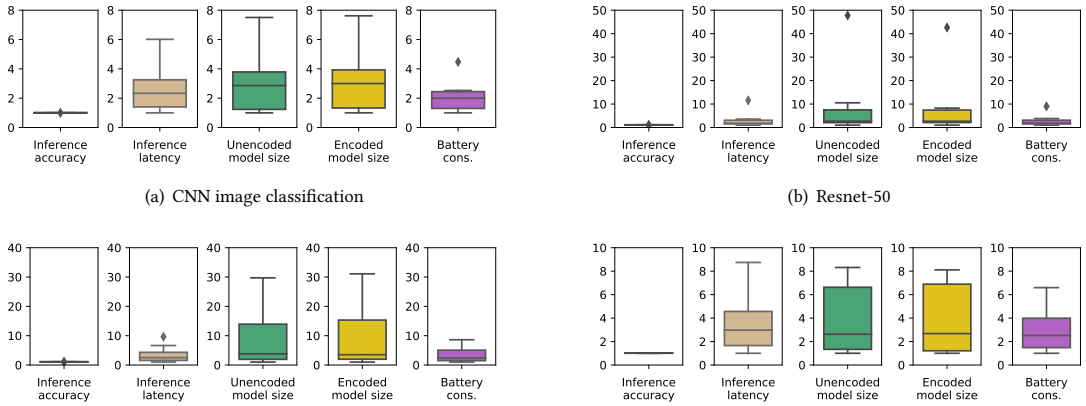
(d) LSTM sentiment classification

Fig. 5. Distribution of the difference between each pair of models' performance property differences before and after model optimization

Table 5. Statistical test results of the comparing the distributions of the pairwise performance property differences between optimized models and standard models.

| Property | CNN image classification | Resnet-50 | CNN text classification | LSTM sentiment classification |
|---|---|---|---|---|
| Inference accuracy | $\ll 0.001$ | $\ll 0.001$ | $\ll 0.001$ | $\ll 0.001$ |
| Inference latency | 0.001 | 0.001 | 0.043 | 0.002 |
| Model size | $\ll 0.001$ | $\ll 0.001$ | $\ll 0.001$ | $\ll 0.001$ |

(i.e., inference accuracy, inference latency, unencoded and encoded model size, and battery consumption) of the best 10 DNN models from tuning all studied hyperparameters after model optimization are summarized in Figure 6. We observe a similar result as in RQ1, which is that, for all our subject models, after applying DNN model optimization methods on the top models, although the models show similar accuracy, they may still have very different performance in terms of other properties (e.g., inference latency or model size). It is worth noting that in addition to the performance properties that are also mentioned in the standard DNN models (cf. Section 4.1), the variance in battery consumption is also remarkable, particularly for the *CNN text classification* model, which has a battery consumption distribution with the first quartile of only 1.50 but a significantly high third quartile of 5.06. The reason behind this finding would be that the state-of-the-art DNN model optimization techniques (e.g., pruning) often aim to optimize the DNN models (i.e., meet computational requirements and reduce model size) for the deployment on resource-constrained devices (e.g., mobile devices or IoT devices), while trying to remain the accuracy of the model. Thus, the optimized DNN models tend to provide very similar accuracy, while other performance properties may have large differences. Therefore, this finding would imply that practitioners should not always choose the optimized DNN model with the best accuracy, since other optimized DNN models (derived from standard model with different hyperparameter configurations) showing slightly




(a) CNN image classification (b) Resnet-50

(c) CNN text classification (d) LSTM sentiment classification

Fig. 6. (Scaled) Performance properties distribution of optimized best-10 DNN models on mobile devices

Note: For each subject model, all the performance properties are normalized by the minimum value of the corresponding property in order to visually compare the distributions among each other.

weaker accuracy may have significantly better improvements in terms of other performance properties (e.g., lower battery consumption).

**The impact of tuning different combinations of hyperparameters on different optimized DNN model performance properties.** Similar to Table 3 in RQ1 (cf. Section 4.1), Table 6 shows the detailed results of comparing the performance of optimized DNN models obtained from tuning all studied hyperparameters with the optimized models from fixing one hyperparameter or fixing one dimension of hyperparameters while tuning all other hyperparameters.

**1) The impact of tuning different combinations of hyperparameters on the performance properties of the optimized DNN models can be different from that of the standard models.** By comparing Table 3 and Table 6, we find that, although the dimensions of hyperparameters that lead to significant differences for inference accuracy are almost identical for both the standard and optimized DNN models for all subject models, the specific hyperparameters that make a difference may be different. For example, while the inference accuracy of the standard *Resnet-50* models is impacted by "Number of epochs" ; as for optimized models, the inference accuracy is significantly influenced by the "Loss function" hyperparameter from the same dimension (i.e., optimizer hyperparameters). In terms of inference latency, the effect of tuning hyperparameters on the standard DNN models and on the optimized ones differ notably. In particular, tuning all three dimensions of hyperparameters have an impact on the inference latency of the standard *CNN image classification* models, while after optimization, only architecture-related hyperparameters cause a significant impact. The different impacts of the hyperparameters on the optimized DNN models and the standard models can be explained by the reason that the impact of the hyperparameter tuning process and the impact of the optimization process are not completely independent. For example, a more complex model resulting from the hyperparameter tuning process may lead to a higher pruning ratio while maintaining similar performance. In addition, the impact of tuning hyperparameters on the encoded and unencoded model size is the same and it remains the same as for the standard models. For instance, before optimizing the *LSTM sentiment classification* model, only one hyperparameter, i.e., "Number of filters/units", leads to a notable impact on the model size, and such impact persists after optimization. After encoding the optimized model, it is still only the "Number of filters/units" that obviously affect the size of the model. Therefore,

based on such a finding, we would suggest that future research should consider the impact of hyperparameter tuning on the performance of optimized DNN models and integrate the model optimization into the hyperparameter tuning process.

**2) Tuning hyperparameters can cause diverse effects on the battery consumption of different optimized DNN models.** With respect to the mobile-specific performance property, i.e., battery consumption, we observe that except for *LSTM sentiment classification models*, our subject DNN models (i.e., *CNN image classification*, *Resnet-50*, and *CNN text classification*) can be significantly impacted by the tuning of the architecture-related hyperparameters. Besides, the optimizer hyperparameters cause notable effects on both the *Resnet-50* model and *LSTM sentiment classification* models while causing an insignificant effect on the *CNN text classification* and *CNN image classification* models. The layer-level model training decision hyperparameters only introduce observable impact on *Resnet-50* models. By further investigating the reason behind this result, we find that the battery consumption is mostly impacted by the architecture-related and optimizer-related hyperparameters since these hyperparameters have a significant impact on the complexity of the models (as they significantly impact the FLOPs performance property); a more complex model tends to be more battery-consuming.

**3) The impact of tuning hyperparameters on the battery consumption and inference latency cannot be measured without running the models on the target devices.** Regarding the performance properties that cannot be measured without running the optimized DNN models on the target device (i.e., battery consumption and inference latency), they can be quite difficult to be integrated into an optimization process. As shown in Table 6, for these performance properties, we find that there exist differences between the hyperparameters influencing battery consumption, inference latency and the hyperparameters influencing other properties. For instance, both layer-level model training decisions and optimizer hyperparameters lead to notable impact on the inference accuracy on *CNN image classification* model, but not on the inference latency, and for *LSTM sentiment classification* model, the hyperparameter "Optimizer" causes significant impact on the battery consumption, but makes no remarkable difference on both unencoded and encoded mode size. Such a finding can be explained by the reason that current hyperparameter tuning techniques (e.g., Keras Tuner) are not able to support objective functions involving performance properties collected from a separate platform, e.g., collecting the runtime performance property (e.g., battery consumption or inference latency) from mobile devices and dynamically feedback to the server-side, thus the impact of tuning hyperparameters on these performance properties may not be considered during the tuning process. Thus, based on this finding, it would be suggested that practitioners need to watch for the impact of the hyperparameters on the battery consumption and inference latency that may not be available without actually running the model on a target device. Our findings also advocate the need for future research on whether to follow a two-step approach (i.e., hyperparameter tuning first and then optimization) or a one-step approach (i.e., hyperparameter tuning and optimization of DNN in the same loop).

Table 6. Overall impact of tuning hyperparameters on the performance of optimized best-10 DNN models on mobile devices

(a) Results for CNN image classification

| Hyperparameter | Inference accuracy | | Inference latency | | Battery consumption | | Unencoded model size | | Encoded model size | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size |
| **Architecture-related hyperparameters** | -0.013 | large | -0.686 | large | -0.548 | large | -0.194 | large | -0.179 | large |
| Number of filters/units | 0.001 | large | -0.048 | large | -0.057 | large | -0.370 | large | -0.299 | large |
| Kernel size | -0.006 | large | -0.187 | large | -0.191 | large | 4.957 | 0.259 | 4.881 | 0.260 |
| **Layer-level model training decisions** | 0.006 | large | 0.387 | 0.285 | 0.359 | 0.396 | 0.919 | 0.213 | 1.063 | 0.154 |
| Dropout ratio | -0.013 | 0.236 | 0.337 | 0.214 | 0.311 | 0.311 | 1.227 | 0.247 | 1.110 | 0.425 |
| Activation function | 0.007 | large | 3.023 | 0.455 | 2.835 | 0.455 | 6.463 | 0.381 | 6.765 | 0.172 |
| Pooling method | 0.007 | 0.381 | 1.713 | 0.137 | 0.806 | 0.052 | 2.767 | 0.366 | 3.012 | 0.260 |
| **Optimizer hyperparameters** | 0.006 | large | 1.738 | 0.425 | 0.903 | 0.285 | 2.767 | 0.236 | 2.926 | 0.121 |
| Loss function | -0.002 | 0.339 | 0.346 | 0.192 | 0.796 | 0.339 | 0.924 | 0.396 | 0.864 | 0.396 |
| Optimizer | -0.001 | 0.236 | 0.030 | 0.285 | -0.019 | 0.137 | $\ll 0.001$ | 0.455 | 0.053 | 0.396 |
| Learning rate | 0.005 | 0.236 | 2.779 | 0.121 | 1.515 | 0.106 | 7.676 | 0.455 | 7.616 | 0.485 |
| Batch size | 0.003 | 0.353 | 1.675 | 0.214 | 1.573 | 0.312 | 2.772 | 0.272 | 2.802 | 0.485 |
| Number of epochs | 0.004 | 0.485 | 1.289 | 0.192 | 1.078 | 0.285 | 9.079 | 0.285 | 8.897 | 0.236 |

(b) Results for Resnet-50

| Hyperparameter | Inference accuracy | | Inference latency | | Battery consumption | | Unencoded model size | | Encoded model size | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size |
| **Architecture-related hyperparameters** | -0.047 | 0.154 | 0.055 | large | 0.024 | large | 2.869 | large | 2.803 | large |
| Number of filters/units | 0.009 | 0.396 | 0.479 | large | 0.244 | large | 2.867 | large | 2.527 | large |
| Kernel size | -0.062 | large | -0.679 | 0.367 | -0.698 | 0.367 | -0.810 | 0.312 | -0.808 | 0.396 |
| **Layer-level model training decisions** | 0.022 | 0.396 | -0.712 | 0.260 | -0.586 | 0.485 | -0.184 | 0.396 | -0.234 | 0.396 |
| Dropout ratio | -0.007 | large | -0.286 | 0.425 | -0.183 | 0.396 | 0.505 | 0.172 | -0.001 | 0.260 |
| Activation function | 0.053 | 0.485 | -0.035 | 0.236 | -0.012 | 0.312 | 0.627 | 0.192 | 0.628 | 0.236 |
| Pooling method | 0.001 | 0.172 | 0.244 | large | 0.220 | large | 2.607 | large | 1.581 | large |
| **Optimizer hyperparameters** | 0.020 | 0.192 | 1.885 | 0.260 | 1.640 | 0.172 | 7.533 | 0.052 | 3.718 | 0.093 |
| Loss function | -0.205 | large | 0.649 | 0.070 | 0.567 | 0.070 | 3.365 | large | 0.879 | large |
| Optimizer | -0.043 | 0.367 | -0.807 | 0.312 | -0.734 | 0.485 | -0.631 | 0.093 | -0.614 | 0.236 |
| Learning rate | 0.041 | 0.106 | -0.047 | 0.339 | 0.024 | 0.260 | 0.635 | 0.106 | 0.580 | 0.121 |
| Batch size | -0.010 | 0.121 | -0.166 | 0.172 | -0.104 | 0.106 | 1.442 | 0.061 | 1.204 | medium |
| Number of epochs | 0.018 | 0.061 | 1.569 | large | 1.530 | large | 4.929 | large | 3.789 | large |

Note: for column "p-value/Effect size", if p-value smaller than 0.05, we present the effect size, else we show the p-value.

Table 6. Overall impact of tuning hyperparameters on the performance of optimized best-10 DNN models on mobile devices

(c) Results for CNN text classification

| Hyperparameter | Inference accuracy | | Inference latency | | Battery consumption | | Unencoded model size | | Encoded model size | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size |
| **Architecture-related hyperparameters** | -0.015 | 0.137 | 0.399 | large | 0.397 | large | -0.453 | large | -0.444 | large |
| Number of filters/units | 0.002 | 0.367 | -0.078 | 0.061 | -0.331 | 0.070 | -0.907 | 0.285 | -0.909 | 0.192 |
| Kernel size | 0.001 | large | 0.578 | 0.052 | 0.483 | 0.070 | 0.026 | 0.172 | 0.046 | 0.154 |
| Embedding dimension | 0.014 | 0.425 | -0.241 | medium | -0.367 | 0.061 | -0.497 | large | -0.511 | large |
| **Layer-level model training decisions** | -0.010 | 0.106 | -0.518 | 0.396 | -0.498 | 0.339 | -0.747 | 0.485 | -0.719 | 0.396 |
| Dropout ratio | -0.004 | 0.172 | -0.518 | 0.367 | -0.498 | 0.339 | -0.922 | 0.485 | -0.922 | 0.396 |
| Activation function | 0.009 | 0.081 | -0.803 | 0.137 | -0.789 | 0.137 | -0.874 | 0.070 | -0.883 | 0.052 |
| Pooling method | 0.011 | 0.081 | 0.889 | 0.285 | 0.310 | 0.312 | -0.686 | 0.260 | -0.678 | 0.192 |
| **Optimizer hyperparameters** | 0.020 | large | -0.295 | 0.367 | -0.475 | 0.367 | -0.866 | 0.339 | -0.861 | 0.285 |
| Loss function | -0.011 | 0.121 | -0.428 | 0.485 | -0.407 | 0.455 | -0.960 | 0.285 | -0.958 | 0.312 |
| Optimizer | -0.009 | 0.192 | 0.378 | 0.367 | 0.371 | 0.485 | -0.716 | 0.260 | -0.702 | 0.285 |
| Learning rate | -0.023 | 0.260 | -0.583 | 0.339 | -0.563 | 0.396 | -0.928 | 0.425 | -0.929 | 0.485 |
| Batch size | -0.027 | 0.214 | -0.256 | 0.455 | -0.445 | 0.455 | -0.921 | 0.339 | -0.918 | 0.285 |
| Number of epochs | -0.011 | 0.070 | 1.215 | 0.137 | 0.603 | 0.192 | -0.638 | 0.093 | -0.593 | 0.061 |

(d) Results for LSTM sentiment classification

| Hyperparameter | Inference accuracy | | Inference latency | | Battery consumption | | Unencoded model size | | Encoded model size | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size | Best model relative diff. | p-value / Effect size |
| **Architecture-related hyperparameters** | 0.023 | 0.470 | 0.466 | 0.106 | 0.438 | 0.070 | -0.485 | 0.234 | -0.488 | 0.260 |
| Number of filters/units | 0.037 | 0.128 | 0.625 | medium | 1.006 | 0.106 | 0.017 | large | 0.002 | large |
| Embedding dimension | 0.020 | 0.136 | 2.860 | 0.455 | 2.874 | 0.367 | -0.428 | 0.154 | -0.451 | 0.154 |
| **Layer-level model training decisions** | 0.001 | 0.154 | 5.463 | 0.172 | 5.450 | 0.339 | 0.111 | 0.236 | 0.100 | 0.172 |
| Dropout ratio | 0.004 | 0.381 | -0.509 | 0.154 | -0.020 | 0.367 | -0.865 | 0.381 | -0.875 | 0.396 |
| Activation function | 0.021 | 0.396 | 2.997 | 0.236 | 3.665 | 0.260 | -0.791 | 0.455 | -0.820 | 0.455 |
| **Optimizer hyperparameters** | 0.005 | 0.396 | 1.574 | 0.121 | 2.509 | 0.172 | 0.045 | 0.410 | 0.013 | 0.396 |
| Loss function | 0.032 | 0.153 | 0.218 | 0.192 | 0.755 | 0.214 | -0.911 | 0.260 | -0.919 | 0.260 |
| Optimizer | 0.004 | large | 0.179 | large | 0.621 | large | -0.482 | 0.106 | -0.490 | 0.106 |
| Learning rate | 0.026 | 0.485 | 3.087 | 0.396 | 2.955 | 0.312 | -0.414 | 0.203 | -0.442 | 0.172 |
| Batch size | 0.018 | 0.099 | -0.657 | 0.121 | -0.341 | 0.121 | -0.870 | 0.106 | -0.877 | 0.106 |
| Number of epochs | 0.015 | 0.425 | -0.739 | 0.106 | -0.483 | 0.137 | -0.873 | 0.312 | -0.879 | 0.339 |

Note: for column "p-value/Effect size", if p-value smaller than 0.05, we present the effect size, else we show the p-value.

Summary of RQ2

We observe that optimization techniques (e.g., pruning) can lead to significantly different performance distributions between the standard DNN models and the optimized ones. For example, for two standard DNN models with similar inference latency on the server, their corresponding optimized models deployed on the mobile device may have very different inference latency. In addition, tuning the hyperparameters may have different impact on the optimized models than on the standard models. Practitioners should consider the impact of model optimization on the performance of optimized models when building and tuning their models in the cloud/server environments. Our findings also advocate the need for future research on whether to follow a two-step approach (i.e., hyperparameter tuning first and then optimization) or a one-step approach (i.e., hyperparameter tuning and optimization of DNN in the same loop).

## 5 DISCUSSION

In this section, we discuss the implications of our empirical study results.

### 5.1 Do not always choose the top-1 DNN model from hyperparameter tuning as the final decision

After submitting many DNN model training jobs with different combinations of hyperparameters, we can acquire a list of generated DNN models prioritized by the target metrics, e.g., accuracy or mean squared error, depending on the target use scenarios of the DNN model. From such a list of models, practitioners often adopt the top-1 DNN model as it has the optimal hyperparameters that yield the best model performance. However, after our comprehensive analysis of the performance of the resulting DNN models, we realize that, compared to other models, although the top-1 DNN model has the best accuracy, it is likely to achieve comparatively worse performance in terms of other aspects. Instead, other candidate models with slightly worse accuracy may outperform the top-1 model in terms of other performance properties. For example, from the results of tuning all the chosen hyperparameters for the *CNN text classification models*, the top-1 model achieves an inference accuracy and an average inference latency of 0.87 and 2,639.26 ms, respectively; the second-best model, while having a trivial degradation in inference accuracy (i.e., 0.86), achieves a significantly faster inference speed (i.e., 967.74 ms). By examining the details of these two DNN models, we find that, compared to the second-best model, the top-1 model has a much more complex model structure (i.e., higher embedding dimensions and more filters) to provide a modest improvement in inference accuracy, but it also causes the model size to be larger and the inference speed to be notably slower.

Therefore, our empirical study results imply that, as current hyperparameter tuning tools (e.g., Keras Tuner or Hyperopt) are mainly based on tuning for one objective (e.g., accuracy or loss), the most accurate DNN model (i.e., top-1) resulting from hyperparameter tuning is not necessarily the most appropriate model. Practitioners should take into account the trade-off between inference accuracy and other important performance properties (e.g., inference latency or mode size) in the specific usage scenarios and context. On the other hand, practitioners can also take into account the multi-objective hyperparameter optimization that achieves an optimal trade-off between various different and even mutually conflicting objectives. For example, in addition to improving the inference accuracy, the objectives regarding inference latency, model size, FLOPs, and/or battery consumption may also be incorporated when choosing the appropriate DNN model in the specific usage scenarios and context (e.g., autonomous vehicles, embedded systems, or mobile phones). There exists much prior work in this field and they propose approaches based on various techniques

such as scalarization [7, 41, 67, 90, 94], Pareto front approximation [28, 48, 58, 80], and decomposition [44, 57, 93] to achieve an optimal trade-off between the individual objectives.

### 5.2 Differences of performance characteristics between standard DNN models for servers/clouds and optimized DNN models for mobile platforms

During the DNN development process, practitioners may face the requirement to deploy a DNN model trained from powerful servers or cloud platforms to resource-constrained mobile devices, e.g., in Android or IOS devices. Due to the fact that a standard DNN model often involves many redundant operations making it complex in structure and large in size, various model optimization methods are adopted to optimize the standard DNN model. However, we find that there exist significant differences between the standard and optimized DNN models in terms of various performance properties. For example, before optimizing the best-10 *CNN image classification models*, the difference in inference latency between these models is not quite significant: the model with the highest inference speed is only 30% faster than the one with the lowest speed. In comparison, after the optimization, the distribution of inference latency becomes highly dispersed: the relative difference of inference speed between the fastest and slowest optimized DNN models increases dramatically to 501% (cf. Table 4).

Hence, the results imply that the performance characteristics (e.g., distribution of inference latency) of standard DNN models for server/cloud platforms may be impacted by the model optimization techniques, thus practitioners need to be careful not to simply transfer the hyperparameter configurations or the understanding of the impact of such hyperparameter settings from one platform to another platform (e.g., tuning hyperparameters on the server and transferring them to mobiles). Instead, one needs to consider performing hyperparameter tuning on the target devices for deployment.

### 5.3 The impact of hyperparameter tuning on performance properties varies across different DNN models

From our empirical study results, we observe that by comparing the performance property distributions of the DNN model generated from tuning and not tuning one hyperparameter or one dimension of hyperparameters, the impact of that hyperparameter or that dimension of hyperparameters on our studied performance properties is not always consistent among various DNN models. For example, in terms of inference latency, tuning the "Dropout ratio" hyperparameter or not leads to significant impact on the *CNN image classification* and *Resnet-50* models, while having trivial effects on the *CNN text classification* and *LSTM sentiment classification* models. The only exception is that, regardless of different DNN models, the "Kernel size" hyperparameter always introduces a notable effect on inference accuracy.

In addition to taking the impact of not tuning a specific hyperparameter into account, we also consider how the interactions among multiple hyperparameters influence the performance properties. In particular, our studied hyperparameters are structured into three dimensions, including architecture-related hyperparameters, layer-level model training decisions, and optimizer hyperparameters. By fixing the value of all hyperparameters within each dimension and comparing the result to fixing each specific hyperparameter in that dimension, we observe the interactions among multiple hyperparameters. For example, from the investigation results of the overall impact of tuning hyperparameters on the performance of standard best-10 DNN models (cf., Table 3), we find that for the *CNN image classification* model, when the values of each hyperparameter in optimizer hyperparameters are fixed, there is no significant effect on inference latency, but when the values of all hyperparameters are fixed, there is a significant effect observed. We also observe such interactions among hyperparameters from the result of optimized standard best-10 DNN models (cf.,

Table 6), for example, for *Resnet-50*, fixing the value of the pooling method causes significant impacts on inference latency, battery consumption, and model size (both unencoded and encoded), while fixing the entire dimensions of hyperparameters (i.e., the layer-level model training decisions), there is no such significant impact on these performance properties.

Our results imply that the impact of a hyperparameter on a certain aspect of a DNN model's performance is not homogeneous across different models and is not independent of other hyperparameters. Instead, there exist interactions among multiple hyperparameters (especially the hyperparameters within the same dimension). These different hyperparameters often influence each other and their impact is related to the structure of the specific DNN models. Practitioners should perform a specific analysis of particular DNN models and performance requirements as the best practices.

## 6 THREATS TO VALIDITY

This section discusses the threats to the validity of our study.

**Construct validity.** One potential threat to the construct validity is the DNN model performance properties used in our experiments. Considering other performance properties, e.g., memory utilization or the number of trained parameters, would benefit our study. However, in order to reduce this threat, we utilize the widely-used and representative properties in practice and literature [10, 34, 47, 53, 86, 87], to evaluate the different performance perspectives of the DNN models. We would also like to note that model size, latency, and FLOPs may be inter-correlated. In fact, we calculated Pearson's correlation for each subject model between these three performance properties, i.e., model size, FLOPs, and inference latency. According to our result, the performance properties are not always correlated, and their correlations can be as low as 0.03. The results can be explained by the fact that these properties provide different and complementary perspectives of DNN performance. For example, although the inference latency of a DNN model would be impacted by FLOPs and model size, it is also affected by the number of memory accesses and other factors. The results prove that these performance properties are not redundant to each other. In addition, the *Resnet-50* model has different blocks; however, during hyperparameter tuning, we opt to regard these different blocks uniformly, which would be a potential threat to the construct validity. The reason for this decision is that *Resnet-50* has 5 different blocks and a total of 48 layers of convolution and 2 layers of pooling; if we treat each of these layers independently, the result for *Resnet-50* is not suitable for horizontal comparison with other models. Besides, our studied hyperparameters include over 11 types of commonly-used hyperparameters in practice, each of which contains a wide range of values, and they already comprise a large hyperparameter search space with a magnitude of $10^{10}$. Thus, we choose to treat each block of *Resnet-50* uniformly. However, more investigations and studies on an even larger search scope (e.g., consider each block of a DNN model independently) are in our ongoing future work. Moreover, we perform the post-training model quantization on the DNN models to convert the precision of weights from 32-bit floats to 8-bit integers. Such optimization operations are by default supported by our used deep learning back-end framework, i.e., TensorFlow, except for the LSTM layer, thus, we do not consider the quantization step on the pruned LSTM models as it is not supported. Due to the lack of readily available techniques for the quantization of LSTM, we could not estimate the impact of skipping this quantization step for LSTM. Therefore, we would leave the investigation of such impacts to our future work.

**Internal validity.** Our study uses the approach of fixing one hyperparameter or fixing one dimension of hyperparameters at a time to understand the impact of tuning different hyperparameters on the performance of the standard and optimized DNN models. However, the choice of the fixed values for the studied hyperparameters or the dimension

of hyperparameters may affect our results. To mitigate this threat, when a hyperparameter needs to be fixed, we use the value adopted in the official example model building code and if the example does not specify the hyperparameter value, we use the default value in the DNN framework API. In addition, when measuring the performance of the DNN models, we perform inference on the testing dataset 30 times and keep the median inference latency result to reduce the noise caused by system warm-up and cool-down. Taking all these measures down to a single value would be a threat to our results. On the other hand, we find that these performance property distributions are rather concentrated, i.e., with an average relative standard deviation (RSD) of 8.03%, and according to prior studies [26, 74], such a small RSD may imply that the performance property distributions are considered low-variance and it would be reasonable to take the median value of all measured results as the final performance property of that model to minimize the noise from the system warm-up and cool-down periods.

**External validity.** Our study is performed on two DNN models for image classification (i.e., *CNN image classification* and *Resnet-50*) and two for text classification (i.e., *CNN text classification* and *LSTM sentiment classification*). However, our study cannot cover all state-of-the-art DNN models (e.g., BERT [27]) and our results may not generalize to them. Under the constraint of computing resources, we consider the four representative DNN models that cover different types of neural networks, including CNN (Convolutional Neural Networks) and RNN (Recurrent Neural Networks) that are widely used in practice [9, 18, 39, 59, 69, 84, 91]. Our results indicate that these DNN models can achieve high accuracy (i.e., the best tuned models of these DNN models achieve 0.77 to 0.99 accuracy). Nevertheless, future work that considers more other DNN models and datasets can benefit our study. In addition, all our studied DNN models are implemented with the TensorFlow deep learning framework, since it is both the most in-demand framework and the fastest growing in recent years [36]. However, different choices of deep learning frameworks may lead to different results. In future work, we will perform experiments on other mainstream deep learning frameworks R2.5 (e.g., PyTorch, Caffe, or MXNet) and other datasets.

## 7 CONCLUSIONS

We noticed that DNN model hyperparameter tuning and optimization techniques are widely adopted by practitioners to ensure the quality of services provided by DNN models. However, how the tuning of different hyperparameters affects a DNN model and its optimized counterpart in terms of various performance properties remains an under-explored area. Improper hyperparameters can lead to sub-optimal models falling to meet desired performance requirements. In this paper, we perform an empirical study of the effect of tuning different DNN model hyperparameters on the standard DNN models and the DNN models that are optimized by pruning, quantization, and encoding, in terms of various performance properties (i.e., inference accuracy, inference latency, model size, FLOPs, and battery consumption). We observe that tuning specific hyperparameters can cause different impact on the performance of DNN models across models and performance properties. Further, model optimization has a confounding effect on the impact of hyperparameters tuning on the model performance. Our findings highlight that practitioners can benefit from paying attention to a variety of performance properties and the confounding effect of model optimization when tuning and optimizing their DNN models. For example, practitioners can improve their choice of the tuned models by choosing one from the top tuned models that has similar accuracy with the most accurate model while possessing significantly better performance in terms of other properties.

This paper provides the following contributions:

- To our best knowledge, this is the first work that comprehensively investigates the relationship between tuning different DNN model hyperparameters and its effect on different performance properties of the standard DNN models and the optimized DNN models.
- We perform comprehensive experiments, tuning up to 11 types of hyperparameters and evaluating the performance of four state-of-the-art DNN models on two platforms (i.e., server and mobile devices) in terms of five performance properties.
- Our results and findings provide insights and guidelines for practitioners who are interested in DNN hyperparameter tuning and DNN model optimization to achieve specific performance requirements, and advocate the need for future research on whether to follow a two-step approach (i.e., hyperparameter tuning first and then optimization) or a one-step approach (i.e., hyperparameter tuning and optimization of DNN in the same loop).

## REFERENCES


[1] 2019. Hyperparameters in Deep Learning. https://towardsdatascience.com/hyperparameters-in-deep-learning-927f7b2084dd. Last accessed 10/10/2020.

[2] 2020. keras code examples. https://github.com/keras-team/keras-io/tree/master/examples. Last accessed 10/16/2020.

[3] 2020. Pruning in Keras example. https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras.

[4] 2020. Tensorflow Model Optimization. https://www.tensorflow.org/model_optimization.

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, and Yuan Yu et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[6] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13, 3 (2017), 1–18.

[7] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M. Kitani. 2018. N2N learning: Network to Network Compression via Policy Gradient Reinforcement Learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[8] Babajide O Ayinde and Jacek M Zurada. 2018. Building efficient convnets using redundant feature pruning. *arXiv preprint arXiv:1802.07653* (2018).

[9] Abdullah Aziz Sharfuddin, Md. Nafis Tihami, and Md. Saiful Islam. 2018. A Deep Recurrent Neural Network with BiLSTM model for Sentiment Classification. In *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*. 1–4.

[10] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2018. Accelerating Neural Architecture Search using Performance Prediction. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net.

[11] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research* 13, 1 (2012), 281–305.

[12] James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013 (JMLR Workshop and Conference Proceedings, Vol. 28)*. JMLR.org, 115–123.

[13] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*. 2546–2554.

[14] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. 2018. Benchmark analysis of representative deep neural network architectures. *IEEE Access* 6 (2018), 64270–64277.

[15] Ekaba Bisong. 2019. Google AutoML: cloud vision. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 581–598.

[16] Carlo Bonferroni. 1936. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze* 8 (1936), 3–62.

[17] Han Cai, Ligeng Zhu, and Song Han. 2018. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *CoRR* abs/1812.00332 (2018).

[18] Jingjing Cai, Jianping Li, Wei Li, and Ji Wang. 2018. Deeplearning Model Used in Text Classification. In *2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*. 123–126.

[19] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. 2016. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678* (2016).

[20] Tse-Hsun Chen, Weiyi Shang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. 2016. Cacheoptimizer: Helping developers configure caching frameworks for hibernate-based database-centric web applications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium*

*on Foundations of Software Engineering*. 666–677.

[21] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A Survey of Model Compression and Acceleration for Deep Neural Networks. *CoRR* abs/1710.09282 (2017).

[22] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. 2020. Universal deep neural network compression. *IEEE Journal of Selected Topics in Signal Processing* (2020).

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

[24] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. 2019. Low-bit Quantization of Neural Networks for Efficient Inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*. IEEE, 3009–3018.

[25] Norman Cliff. 2014. *Ordinal methods for behavioral data analysis*. Psychology Press.

[26] Mauricio Farias Couto, Luiz Alexandre Peternelli, and Márcio Henrique Pereira Barbosa. 2013. Classification of the coefficients of variation for sugarcane crops. *Ciência rural* 43 (2013), 957–961.

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186.

[28] Michael T. M. Emmerich, André H. Deutz, and Jan Willem Klinkenberg. 2011. Hypervolume-based expected improvement: Monotonicity properties and exact computation. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011, New Orleans, LA, USA, 5-8 June, 2011*. IEEE, 2147–2154.

[29] Livia Faes, Siegfried K Wagner, Dun Jack Fu, Xiaoxuan Liu, Edward Korot, Joseph R Ledsam, Trevor Back, Reena Chopra, Nikolas Pontikos, Christoph Kern, et al. 2019. Automated deep learning design for medical image classification by health-care professionals with no coding experience: a feasibility study. *The Lancet Digital Health* 1, 5 (2019), e232–e242.

[30] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 1436–1445.

[31] Chris Fawcett and Holger H. Hoos. 2016. Analysing differences between algorithm configurations through ablation. *J. Heuristics* 22, 4 (2016), 431–458.

[32] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-Sklearn 2.0. *arXiv:2007.04074 [cs.LG]* (2020).

[33] Yanjie Gao, Yu Liu, Hongyu Zhang, Zhengxian Li, Yonghao Zhu, Haoxiang Lin, and Mao Yang. 2020. *Estimating GPU Memory Consumption of Deep Learning Models*. Technical Report MSR-TR-2020-20. Microsoft.

[34] Robert B Gramacy, Matt Taddy, and Stefan M Wild. 2013. Variable selection and sensitivity analysis using dynamic trees, with an application to computer code performance tuning. *The Annals of Applied Statistics* (2013), 51–80.

[35] Jia Guo and Miodrag Potkonjak. 2017. Pruning convnets online for efficient specialist models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 113–120.

[36] Jeff Hale. 2019. Which Deep Learning Framework is Growing Fastest? https://towardsdatascience.com/which-deep-learning-framework-is-growing-fastest-3f77f14aa318.

[37] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[38] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. *CoRR* abs/1506.02626 (2015).

[39] Seong-Hyeon Han and Kwang-Yeob Lee. 2017. Implemetation of image classification CNN using multi thread GPU. In *2017 International SoC Design Conference (ISOCC)*. 296–297.

[40] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866* (2018).

[41] Chi-Hung Hsu, Shu-Huan Chang, Da-Cheng Juan, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Shih-Chieh Chang. 2018. MONAS: Multi-Objective Neural Architecture Search using Reinforcement Learning. *CoRR* abs/1806.10332 (2018).

[42] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Identifying Key Algorithm Parameters and Instance Features Using Forward Selection. In *Learning and Intelligent Optimization - 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 7997)*, Giuseppe Nicosia and Panos M. Pardalos (Eds.). Springer, 364–381.

[43] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2014. An Efficient Approach for Assessing Hyperparameter Importance. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014 (JMLR Workshop and Conference Proceedings, Vol. 32)*. JMLR.org, 754–762.

[44] Jing Jiang, Fei Han, Qinghua Ling, Jie Wang, Tiange Li, and Henry Han. 2020. Efficient network architecture search via multiobjective particle swarm optimization based on decomposition. *Neural Networks* 123 (2020), 305–316.

[45] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar,

Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 1946–1956.

[46] Julie. 2020. Hands on hyperparameter tuning with Keras Tuner. https://www.sicara.ai/blog/hyperparameter-tuning-keras-tuner.

[47] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530* (2015).

[48] Joshua D. Knowles. 2006. ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Trans. Evol. Comput.* 10, 1 (2006), 50–66.

[49] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *J. Mach. Learn. Res.* 18 (2017), 25:1–25:5.

[50] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[52] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[53] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. 2016. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In *2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom)*. IEEE, 477–484.

[54] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

[55] Lizhi Liao, Jinfu Chen, Heng Li, Yi Zeng, Weiyi Shang, Jianmei Guo, Catalin Sporea, Andrei Toma, and Sarah Sajedi. 2020. Using black-box performance models to detect performance regressions under varying workloads: an empirical study. *Empirical Software Engineering* 25, 5 (2020), 4130–4160.

[56] Marius Lindauer and Frank Hutter. 2019. Best Practices for Scientific Research on Neural Architecture Search. *CoRR* abs/1909.02453 (2019).

[57] Jia Liu, Maoguo Gong, Qiguang Miao, Xiaogang Wang, and Hao Li. 2018. Structure Learning for Deep Neural Networks Based on Multiobjective Optimization. *IEEE Trans. Neural Networks Learn. Syst.* 29, 6 (2018), 2450–2463.

[58] Mohammad Loni, Sima Sinaei, Ali Zoljodi, Masoud Daneshtalab, and Mikael Sjödin. 2020. DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocess. Microsystems* 73 (2020), 102989.

[59] Yuandong Luan and Shaofu Lin. 2019. Research on Text Classification Based on CNN and LSTM. In *2019 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. 352–355.

[60] Yuexin Ma, Xinge Zhu, Sibo Zhang, Ruigang Yang, Wenping Wang, and Dinesh Manocha. 2019. Trafficpredict: Trajectory prediction for heterogeneous traffic-agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 6120–6127.

[61] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 142–150.

[62] Abhinav Mehrotra, Alberto Gil C. P. Ramos, Sourav Bhattacharya, Lukasz Dudziak, Ravichander Vipperla, Thomas C. P. Chau, Mohamed S. Abdelfattah, Samin Ishtiaq, and Nicholas Donald Lane. 2021. NAS-Bench-ASR: Reproducible Neural Architecture Search for Speech Recognition. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

[63] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[64] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440* (2016).

[65] Nadim Nachar et al. 2008. The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution. *Tutorials in quantitative Methods for Psychology* 4, 1 (2008), 13–20.

[66] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. 2019. Keras Tuner. https://github.com/keras-team/keras-tuner.

[67] Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. 2019. A Flexible Framework for Multi-Objective Bayesian Optimization using Random Scalarizations. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019 (Proceedings of Machine Learning Research, Vol. 115)*, Amir Globerson and Ricardo Silva (Eds.). AUAI Press, 766–776.

[68] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

[69] Saeed Mian Qaisar. 2020. Sentiment Analysis of IMDb Movie Reviews Using Long Short-Term Memory. In *2020 2nd International Conference on Computer and Information Sciences (ICCIS)*. 1–4.

[70] Elad Rapaport, Oren Shriki, and Rami Puzis. 2019. EEGNAS: Neural Architecture Search for Electroencephalography Data Analysis and Decoding. In *Human Brain and Artificial Intelligence - First International Workshop, HBAI 2019, Held in Conjunction with IJCAI 2019, Macao, China, August 12, 2019, Revised Selected Papers (Communications in Computer and Information Science, Vol. 1072)*, An Zeng, Dan Pan, Tianyong Hao, Daoqiang Zhang, Yiyu Shi, and Xiaowei Song (Eds.). Springer, 3–20.

[71] Nils Reimers and Iryna Gurevych. 2017. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *arXiv preprint arXiv:1707.06799* (2017).

[72] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2020. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *CoRR* abs/2006.02903 (2020).

[73] C Saranya and G Manikandan. 2013. A study on normalization techniques for privacy preserving data mining. *International Journal of Engineering and Technology (IJET)* 5, 3 (2013), 2701–2704.

[74] Macfarlane TU Scott, Tannath J Scott, and Vincent G Kelly. 2016. The validity and reliability of global positioning systems in team sport: a brief review. *The Journal of Strength & Conditioning Research* 30, 5 (2016), 1470–1490.

[75] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[76] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958.

[77] James H Stapleton. 2007. *Models for probability and statistical inference: theory and applications*. Vol. 652. John Wiley & Sons.

[78] Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. 2021. Training with Quantization Noise for Extreme Model Compression. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

[79] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[80] El-Ghazali Talbi. 2019. A unified view of parallel multi-objective evolutionary algorithms. *J. Parallel Distributed Comput.* 133 (2019), 349–358.

[81] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2820–2828.

[82] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy (Eds.). ACM, 847–855.

[83] Frederick Tung and Greg Mori. 2018. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7873–7882.

[84] An Tien Vo, Hai Son Tran, and Thai Hoang Le. 2017. Advertisement image classification using convolutional neural network. In *2017 9th International Conference on Knowledge and Systems Engineering (KSE)*. 197–202.

[85] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. 2016. Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*. 606–615.

[86] Jenna Wong, Travis Manderson, Michal Abrahamowicz, David L Buckeridge, and Robyn Tamblyn. 2019. Can hyperparameter tuning improve the performance of a super learner?: A case study. *Epidemiology (Cambridge, Mass.)* 30, 4 (2019), 521.

[87] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. 2018. Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. 278–291.

[88] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W. Mahoney, and Kurt Keutzer. 2021. HAWQ-V3: Dyadic Neural Network Quantization. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 11875–11886.

[89] Shaokai Ye, Xiaoyu Feng, Tianyun Zhang, Xiaolong Ma, Sheng Lin, Zhengang Li, Kaidi Xu, Wujie Wen, Sijia Liu, Jian Tang, et al. 2019. Progressive dnn compression: A key to achieve ultra-high weight pruning and quantization rates using admm. *arXiv preprint arXiv:1903.09769* (2019).

[90] Min Yoon, Yeboon Yun, and Hirotaka Nakayama. 2009. *Sequential Approximate Multiobjective Optimization Using Computational Intelligence*. Springer.

[91] Zharfan Zahisham, Chin Poo Lee, and Kian Ming Lim. 2020. Food Recognition with ResNet-50. In *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*. 1–5.

[92] Julie Zelenski, K. Huffman, K. Schwarz, and Marty Stepp. 2012. Huffman Encoding and Data Compression.

[93] Qingfu Zhang and Hui Li. 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Trans. Evol. Comput.* 11, 6 (2007), 712–731.

[94] Richard Zhang and Daniel Golovin. 2020. Random Hypervolume Scalarizations for Provable Multi-Objective Black Box Optimization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 11096–11105.