The Art of Getting Deep Neural Networks in Shape

RAHIM MAMMADLI, Technische Universität Darmstadt, Graduate School of Excellence Computational Engineering, Germany
FELIX WOLF, Technische Universität Darmstadt, Department of Computer Science, Germany
ALI JANNESARI, Iowa State University, Department of Computer Science, USA

Training a deep neural network (DNN) involves selecting a set of hyperparameters that define the network topology and influence the accuracy of the resulting network. Often, the goal is to maximize prediction accuracy on a given dataset. However, non-functional requirements of the trained network – such as inference speed, size, and energy consumption – can be very important as well. In this article, we aim to automate the process of selecting an appropriate DNN topology that fulfills both functional and non-functional requirements of the application. Specifically, we focus on tuning two important hyperparameters, depth and width, which together define the shape of the resulting network and directly affect its accuracy, speed, size, and energy consumption. To reduce the time needed to search the design space, we train a fraction of DNNs and build a model to predict the performances of the remaining ones. We are able to produce tuned ResNets, which are up to 4.22 times faster than original depth-scaled ResNets on a batch of 128 images while matching their accuracy.

CCS Concepts: • **Computing methodologies → Neural networks**; *Supervised learning*;

Additional Key Words and Phrases: Deep neural networks, computer vision, parallel processing

ACM Reference format:
Rahim Mammadli, Felix Wolf, and Ali Jannesari. 2019. The Art of Getting Deep Neural Networks in Shape. *ACM Trans. Archit. Code Optim.* 15, 4, Article 62 (January 2019), 21 pages.
https://doi.org/10.1145/3291053

1 INTRODUCTION

Deep neural networks (DNNs) represent effective solutions to problems across several different domains, including computer vision. One of the most popular tasks in this field is object classification, in which a given input image must be assigned to the class best representing what is displayed on that image. The best results for this task have been achieved by convolutional neural networks, with LeNet [18] being the first to recognize handwritten digits from the MNIST dataset with low error in 1998 and AlexNet [15] substantially outperforming the state of the art in the large-scale image classification task ImageNet [13] in 2012. Since then, several prominent DNN architectures

This work is a new article, not an extension of a conference paper.
Authors' addresses: R. Mammadli, Technische Universität Darmstadt, Graduate School of Excellence Computational Engineering, 64293, Darmstadt, Hessen, Germany; email: mammadli@cs.tu-darmstadt.de; F. Wolf, Technische Universität Darmstadt, Department of Computer Science, 64293, Darmstadt, Hessen, Germany; email: wolf@cs.tu-darmstadt.de; A. Jannesari, Iowa State University, Department of Computer Science, 50011-1090, Ames, IA, USA; email: jannesari@iastate.edu.

1544-3566/2019/01-ART62
https://doi.org/10.1145/3291053

have emerged, including residual networks – ResNets [9] and densely connected convolutional networks – DenseNets [11], with current state-of-the-art results shown by machine-engineered NASNets [31]. We take inspiration from wide residual networks [28], which introduced the parameter of width in their work and demonstrated the potential of scaling ResNets by increasing their width as opposed to their depth. We are specifically interested in wide and shallow architectures, since they introduce opportunities to exploit layer-wide parallelism and reduce the number of sequential computational steps required to process any given input. While our work concentrates on exploring the design space of residual DNNs (i.e., ResNets), we believe that it is possible to achieve similar results using other DNN architectures, such as DenseNets and NASNets.

Even for a fixed neural network architecture, the design space consisting of networks of different width and depth can be quite large. For a given application with requirements not only on accuracy but also size, speed, and energy consumption of a neural network, the choice of the optimal DNN topology is far from trivial. We tackle this problem by developing a library of tools under the umbrella name of Shape_DNN that automates the process of choosing the optimal pair of values for depth and width for a given classification problem. Shape_DNN can measure the speed and energy consumption of DNNs and predict their accuracy based on the classification performance of a subset of shapes in the design space. DNNs in the design space are initialized with random weights to measure their speed and energy. Shape_DNN trains a subset of DNNs in the design space and models their accuracy, employing one of several predefined modeling strategies, to predict the accuracy of the rest. Hence, it becomes possible to answer the following questions:

- Which is the most accurate shape for a DNN that does not exceed a given maximum size?
- Which is the fastest shape for a DNN that meets a certain minimum accuracy?
- Which shape is the most energy efficient that does not fall below a given minimum accuracy and does not exceed a given maximum size?
- Which shapes meet the requirements on speed and size while performing with accuracy above a certain threshold?

Our experiments show that Shape_DNN can produce DNNs that are smaller, faster, and more energy efficient than the original depth-scaled ResNet implementations introduced by He et al. [10] without sacrificing accuracy. For example, we have produced a DNN that is 4.22 times faster than the baseline defined by He et al. on a batch of 128 images while achieving the same level of test-set accuracy with the CIFAR-10 dataset.

The structure of the article is as follows. In Section 2, we discuss related work and highlight differences to ours. In Section 3, we explain the relationship between the shape of a neural network and its non-functional requirements and present our approach to measuring the energy consumption of DNNs on NVIDIA GPUs. Then, we present and motivate the strategies employed by Shape_DNN for searching the design space of DNNs in Section 4. We evaluate our approach using four datasets in Section 5. We conclude with Section 6, in which we review our approach and consider possible future work.

2 RELATED WORK

In practice, the process of hyperparameter optimization of DNNs is often done manually or by using grid search [17]. Training a DNN is a time-consuming task and a scientist with expert knowledge can help narrow the search space. On the other hand, grid search is attractive because of its ease of parallel implementation, which makes it possible for many DNN architectures to be trained simultaneously. Random search, as opposed to grid search, was shown to be more efficient when searching large design spaces [3].

In recent years, several hyperparameter-optimization methods have been proposed that compete with random search on this task. In general, the difference between Shape_DNN and these approaches is the scope of the target problem. Shape_DNN tunes a *given* DNN architecture, while the other methods that we discuss below try to design a new DNN architecture. Another difference in Shape_DNN is that it considers size, speed, and energy efficiency of the DNN along with its accuracy.

Miikkulainen et al. [20] propose an automated method for optimizing a DNN architecture using evolutional algorithms. They define a pool of hyperparameters that produce an exponentially larger design space than the one considered by Shape_DNN. Therefore, the amount of training performed by Shape_DNN and its computational demands are much smaller. To reduce the computational complexity of their approach, Miikkulainen et al. [20] train the population of DNNs for only a few epochs, which, as mentioned by the authors, results in a bias toward fast-learning architectures. Indeed, in our experiments wider shapes tend to learn much faster than their deeper alternatives of the same size. However, deeper shapes tend to learn more in later epochs. We therefore train DNNs to convergence and then use their test-set accuracy to build Shape_DNN's model for predicting accuracy. Since the two approaches focus on solving different problems, they can be used together: one to produce an architecture and the other to tune it to meet the application requirements on a given hardware.

Zoph and Le [30] use a controller recurrent neural network (RNN) architecture to predict variable-sized specifications of designs of DNN architectures that have the best validation-set accuracy. In contrast to the work of Miikkulainen et al. [20], which uses a neuroevolution algorithm to search for the best-performing architecture, Zoph and Le use reinforcement learning to train the RNN, which allows the use of gradient-based optimization of the predicting model. As in the work discussed previously, the goal is to design an architecture, although the design spaces considered are smaller; for example, the authors use predefined values for the learning rate, weight decay, momentum, and so on, instead of learning them. Nevertheless, this design space is still exponentially larger than the one explored in our work. Since the authors' goal is training a controller RNN, the accuracy of which depends largely on the amount of training data, and since the design space explored is very large, their computational requirements are demanding. Specifically, the authors used a distributed setting with 100 nodes, each equipped with 8 GPUs, to train a total of 12,800 different DNN architectures on a CIFAR-10 classification task. In contrast, in our experiments with the same dataset, we used a single desktop with one GPU. The biggest design space that we considered consisted of 180 different shapes; we trained only 45 and used our model to predict the performances of the remaining ones. Given that our linear regression model is much simpler than the RNN employed by the authors, it also requires a much smaller amount of training data to achieve good prediction accuracy. Nevertheless, we considered shape sets with a larger variety of widths than the authors (20 vs. 4). This allowed us to fine-tune a ResNet architecture to produce models that fit various constraints.

Inspired by the success of recent scalable architectures, such as ResNet and DenseNet, Zoph et al. [31] use reinforcement learning to produce two types of cells used to build a DNN architecture with the goal of maximizing validation-set accuracy on CIFAR-10. Normal cells preserve the size of the input channels and reduction cells reduce their width and depth. The size of the design space explored by the authors, although smaller than the one in the previous work, is still exponentially larger than the space considered by Shape_DNN. The experiments on CIFAR-10 were run in parallel on 500 GPUs for 4 days to determine the most accurate cells. The result of their work is the NAS cell, which is used to build the NASNet architecture, showing state-of-the-art results on both CIFAR-10 and ImageNet classification tasks. The authors parameterize the NASNet architecture with the number of cell repeats, whose effect is similar to the depth parameter of wide ResNets, and the number of filters in the first layer of the network, which also determines the number of filters in subsequent layers of the network and whose effect is similar to the width parameter of wide ResNets. However, the authors use exhaustive grid search instead of accuracy models to search this space, which is very expensive and not practical for many real-life applications of DNNs. Therefore, our approach to modeling accuracy—based on the shape of the network—can significantly speed up this stage of architecture search by estimating the accuracy of the networks, rather than training all of them. In the future, we plan to add support for NASNets to Shape_DNN.

Negrinho and Gordon [21] propose a language for defining the design space of DNNs and a framework for efficiently searching this space. The results obtained on CIFAR-10 show that their approach outperforms random search. However, the DNNs produced are less accurate than other state-of-the-art models. Similar to Zoph and Le [30], Baker et al. [2] use reinforcement learning to produce DNN architectures. However, the design spaces that they consider are smaller when compared to the ones explored by Zoph and Le. They build DNNs using only standard convolutional, fully connected, and pooling layers. Therefore, the DNNs that they produce are less accurate than architectures such as ResNet, which use skip connections and identity mappings.

Zagoruyko and Komodakis [28] trained residual networks of various widths and compared their accuracy when applied to several datasets. We extend their work by modeling the accuracy, measuring the speed and energy consumption, and calculating the size of DNNs of various shapes and use this information to efficiently navigate the design space. There exists a body of research on performance modeling of DNNs. For example, PALEO [22] is an analytical performance model that estimates training and inference times of a DNN based on its declarative specification. As opposed to PALEO, we focus on tuning the DNN shapes with the aim of minimizing inference times rather than training times. However, we also consider other characteristics, most notably accuracy, because we believe that both speed and accuracy of the resulting network must be considered together for production of fine-tuned DNNs. Hashemi et al. [8] analyze the scalability of training DNNs using the CNTK framework on various configurations of a cluster. However, they do not model the accuracy of DNNs. To the best of our knowledge, there has been no research so far considering all of the abovementioned properties of a DNN at the same time.

Several approaches related to designing energy-efficient DNNs have been proposed in recent years. There is a substantial amount of literature on achieving energy efficiency by reducing the size of a DNN. The proposed methods include reducing the number of parameters of a DNN by using a different architecture [12], compressing the network [7], and using low-precision weights [29], including binary weights [6] and activations [23]. Binarization techniques offer additional advantages. Using binary weights and activations, it is possible to implement most of the computation in a DNN with accumulate [6] or even XNOR [23] instructions, which are faster and more energy efficient than regular multiply-accumulate instructions. Yang et al. use hardware measurements for the energy-aware pruning of parameters [27]. We ran our experiments and measured energy on NVIDIA GPUs rather than resorting to a specialized hardware, as Yang et al. do. However, our results confirm their findings that estimating energy based only on the number of parameters of a DNN is inaccurate, as it ignores other important properties, such as the network topology.

3 DESIGN SPACE OF DNN SHAPES

In this section, we explore the relationship between depth and width of a DNN on the one hand and its non-functional properties on the other, including inference speed, memory requirements, and energy consumption. In contrast to purely theoretical considerations [12, 24], we illustrate these relationships experimentally under the constraints of actual and finite GPU hardware. Our aim is to provide the reader with intuition about dependence between the non-functional requirements

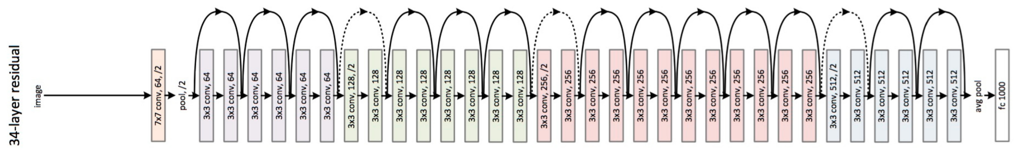


Fig. 1. Example of a ResNet architecture (adapted from He et al. [9] with permission). Curved lines represent addition operations within residual blocks.

and the shape of a DNN and to illustrate the kinds of computations performed by Shape_DNN for the networks in the design space. First, we briefly describe the ResNet architecture and our training datasets, followed by the setup of our test environment, before we introduce the terminology used throughout this article and analyze each non-functional requirement in detail.

3.1 ResNet Architecture

The main contribution of the ResNet architecture is the introduction of a residual block consisting of several convolutional layers. Each block adds its inputs to the output of the last convolutional layer. This technique makes it possible to train very deep architectures and prevents accuracy degradation caused by the increased depth of the network. Figure 1 displays an example of a ResNet architecture. Zagoruyko and Komodakis [28] showed with wide residual networks that, in certain scenarios, scaling networks by increasing their width instead of depth can more easily reach higher prediction accuracy. The measurements presented in this section stem from wide ResNet topologies. However, the dependences are valid for other types of convolutional neural networks as well.

3.2 Datasets

We trained various ResNet topologies on four different datasets: MNIST [19], CIFAR-10, CIFAR-100 [14] and ImageNet32×32 [5]. MNIST contains gray-scale images of dimension $28 \times 28 \times 1$ with training and test sets consisting of 60,000 and 10,000 images, respectively. CIFAR-10 and CIFAR-100 consist of 60,000 labeled images of dimensions $32 \times 32 \times 3$, of which 50,000 were used for training and 10,000 for testing. The images are divided into 10 and 100 classes, respectively. ImageNet32×32 is a downsampled version of the ImageNet [13] dataset and consists of the same images but reduced to dimensions $32 \times 32 \times 3$. Because downsampling eliminates information, it presents a harder classification task than the original ImageNet but is at the same time computationally less demanding and, therefore, more amenable to experimental studies. The dataset consists of 1,281,167 training and 50,000 test images from 1000 classes.

3.3 Test Environment

Our measurements and training for MNIST and CIFAR datasets have been performed using the PyTorch [25] framework on a single desktop computer with an Intel(R) Core(TM) i7-4770 CPU, 32GB of main memory, and two GPUs, one NVIDIA TITAN Xp and one NVIDIA GeForce GTX TITAN X, and the Ubuntu 16.04 LTS operating system. The results presented for these datasets were collected using the NVIDIA Titan Xp; the second GPU was used only for cross-checking the results and measurements. To speed up the design space exploration of ImageNet32×32, we trained our models on accelerator-equipped nodes of Lichtenberg, an HPC cluster at TU Darmstadt. Each of the nodes contains two NVIDIA Tesla K20 GPUs. The experiments on this dataset, including measurements of energy consumption and speed of DNNs, were performed on two servers: one with a single NVIDIA Tesla K20 GPU and another with a single NVIDIA Tesla K20Xm GPU.

3.4 Terminology

We will further refer to the amount of time it takes to process an input by a DNN as its inference time or *latency*. For the sake of consistency, we adopt the definitions of width and depth from Zagoruyko and Komodakis [28]. *Depth* is defined as the number of convolutional layers and *width* is the factor used to multiply the number of channels in each convolutional layer of the network. The width of 1 corresponds to the original ResNet implementation by He et al. [9]. We refer to a combination of depth and width as the *shape* of a DNN architecture and will be referring to various shapes in a *(<depth>, <width>)* format. We extend the work of Zagoruyko and Komodakis [28] by including *thin* residual networks into the design spaces that we consider, which, as opposed to being widened, are shrunk to reduce the size of the network.

3.5 Size of the DNN

The size of a wide ResNet topology, which determines its memory consumption, can be calculated based on four parameters:

- the size of an input
- the number of output classes
- depth
- width

The first two parameters are defined by the classification problem and are therefore not considered for tuning. The relation between the size and the depth of the network is linear, whereas the size grows proportionally with the square of the width. This is because convolutional layers are four-dimensional and increasing the width increases both the number of input and output channels equally for each convolutional layer.

3.6 Latency

The architectural specification of a DNN also defines its computational graph. For convolutional neural networks, which are the state of the art for object classification tasks, most of the computation resides in convolutional blocks. Although deeper neural network architectures have been shown to be able to approximate more complex functions than shallow ones, they come with an increased computational cost. Specifically, the increase in the number of convolutional blocks results in higher network latency. Latency requirements, for example, in real-time scenarios, might limit the applicability of such deep architectures in practice.

Considering that the majority of the state-of-the-art DNNs are being trained and run on GPUs, which are designed to handle embarrassingly parallel computations while not being equally well suited for the execution of sequential code, shallow and wide DNN architectures have the potential to yield faster inference and training times as opposed to their deep and thin alternatives.

The amount of work required to feed-forward an input through a DNN increases quadratically with the width and linearly with the depth. However, despite the work being asymptotically dominated by the width, the amount of concurrency per layer might still be less than what a GPU can provide. For example, if we compare the two Shapes A and B displayed in Figure 2, we can observe that A is wider and shallower than B and requires more work to forward propagate an input. However, despite this, it has a lower latency because it can exploit a higher degree of parallelism.

To demonstrate this relationship, we have constructed and measured the latencies of various shapes of ResNets with batch sizes from 1 up to 128. As expected, the latency increases linearly with depth, as can be seen in Figure 3(a).

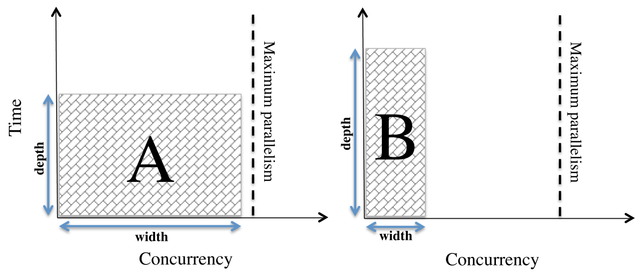


Fig. 2. Shape A has less latency than shape B, despite requiring more work (area = size of the shape), because it can use parallel hardware more efficiently.

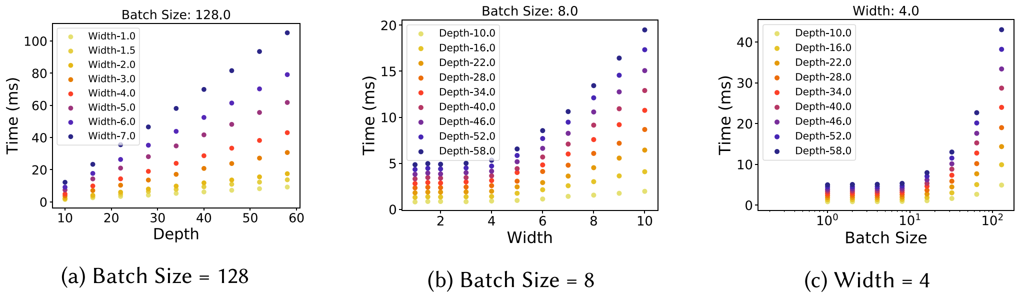


Fig. 3. From left to right, relation between latency (ms) and the depth, width, and batch size of the network. Batch sizes are on a logarithmic scale.

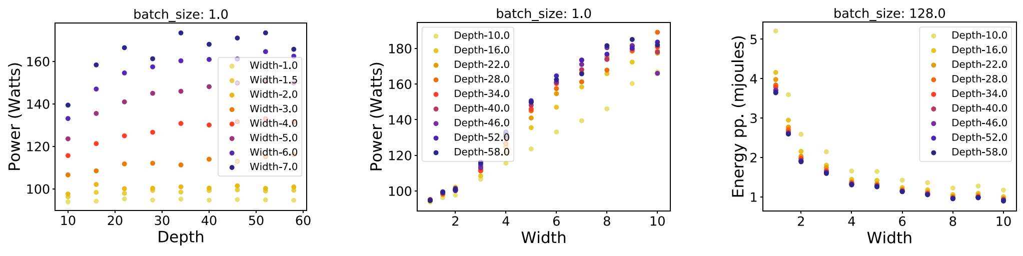
Wider shapes, however, despite having quadratically more parameters and hence overall computation, result in a quadratic increase in latency only after a certain threshold is reached. This is likely because, up to this point, they can exploit layer-wide parallelism, letting increasing width result in higher GPU utilization but not in higher latency. Only after the potential parallelism exceeds what the hardware can support, grows the latency with the square of the width. Figure 3(b) shows very slight changes in latency for the shapes of different widths and a batch size of 8. As can be seen after a certain threshold, a width of 4 in this case, latency starts increasing quadratically. The quadratic relationship between latency and the width of the network also explains why the latency scales less dramatically for shapes of width 1.0, 1.5, and 2.0 than for others, as can be seen in Figure 3(a). Once the GPU is fully utilized, all the parallelism of the hardware available to speed up the computation of individual layers is exhausted and any additional computation will introduce serialization within the layer.

Similarly, increasing the batch size for thinner topologies results in higher GPU utilization and, after a certain threshold, increases the latency linearly, as can be observed in Figure 3(c). This is because processing larger batches also results in an increase of the amount of work per layer.

GPU utilization can be viewed as a function of width and batch size. By tuning these values together, it is possible to ensure efficient use of GPU resources.

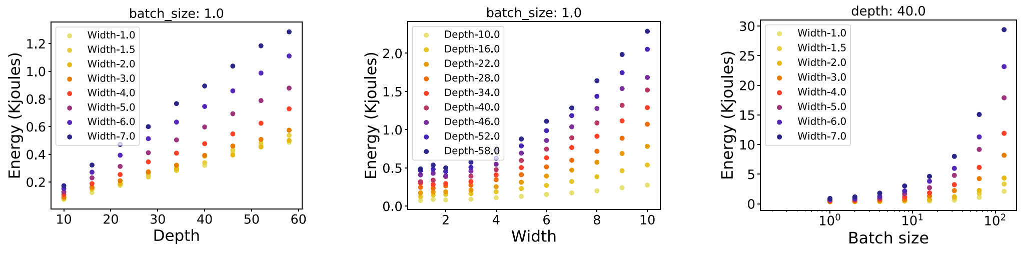
3.7 Power Use and Energy Consumption

Most modern NVIDIA GPUs have on-chip sensors measuring power use, which are accessible through the NVIDIA Management Library (NVML) interface. Our approach to measuring energy consumption of various DNN topologies is based on recording actual power measurements provided by these sensors. Similar approaches [4, 16] have been used to calculate the energy



(a) Batch size = 1 (b) Batch size = 1 (c) Batch size = 128

Fig. 4. The figure on the left and the figure in the center display the relation between power usage and the depth and width of a DNN, respectively. On the right is the dependence between energy consumption per parameter and width.



(a) Batch size = 1 (b) Batch size = 1 (c) Depth = 40

Fig. 5. From left to right, relation between energy consumption and the depth, width, and the batch size. Batch sizes are on a logarithmic scale.

consumption of various kernels. To measure the energy consumption of a set of topologies, we spawn two processes, one of which performs forward-propagation and the other records power draw until the point when it stabilizes or a time-out is reached. The mean power draw and latency values are used to calculate the energy consumption for each topology.

We measured the power use and energy consumption of ResNets with various shapes. As is evident from Figure 4(b), the power use increases linearly with the width until the power limit of the device is reached. This is because wider networks with their higher degree of layer-wider parallelism use the GPU to a larger extent. Increasing the depth, however, has no effect on the amount of parallelism per layer and, therefore, has a very small impact on power use, as shown in Figure 4(a).

Increasing the depth of a DNN will linearly increase the energy consumption, as shown in Figure 5(a). The reason is that deeper topologies have the same amount of work per layer and use as much power but for a longer period of time. Increasing width, on the other hand, initially expands the network into previously unused regions of the GPU, which, however, had to be powered anyway – albeit statically. After the GPU is fully occupied and no more hardware concurrency can be exploited, energy grows proportionally with the amount of work, which raises quadratically with increasing width, as can be seen in Figure 5(b).

As a consequence of this relationship, the amount of energy consumed per parameter is higher when the GPU is underutilized, as can be observed in Figure 4(c). Once the GPU is fully utilized, the extra energy consumed for every additional parameter is the same. For this reason, wider shapes can sometimes be more energy efficient than their thinner deeper alternatives while being bigger in size. For example, DNN shape (16, 5) needs 2.1 times less energy to feed forward a single image than (58, 1) while being 5 times larger. As stated earlier, our observations agree with the results achieved by Yang et al. [27] in that reducing the number of parameters of a DNN does not always improve energy efficiency. Similarly, increasing the batch size will linearly increase energy consumption after no more batch-level parallelism can be exploited, which can be seen in Figure 5(c). Note that the batch size in the figure is scaled logarithmically.

4 SEARCHING THE DESIGN SPACE

In this section, we describe the search algorithm of Shape_DNN. As input, Shape_DNN expects the specification of the design space, training, and test datasets for a given classification task, a set of constraints, and, optionally, an optimization objective. Moreover, the user can choose among three modeling strategies, which will be discussed below. Constraints can be imposed on any combination of size, latency, energy consumption, and accuracy. The optimization objective can be chosen among these four criteria. It is also possible to modify the default values for other hyperparameters, such as the learning rate, and so on. The output of the algorithm is the set of DNN shapes complying with the provided constraints or, if the optimization goal is present, the best shape out of this set with respect to the optimization objective.

First, the design space of DNNs is defined based on user-supplied ranges of values for depth and width, specified in terms of minimum, maximum, and step size. The space is then filtered based on the chosen constraints, starting with size, followed by latency, energy, and, finally, test-set accuracy in that order. DNNs are first filtered based on their size because the size of a DNN can be derived quickly as a function of the shape without instantiating the network. Next, Shape_DNN measures the latency of the remaining shapes after initializing them with random weights and eliminates those exceeding the desired latency limit, if there is one. The survivors are subjected to energy tests and possibly filtered based on energy constraints.

After all non-functional criteria have been applied, Shape_DNN selects a subset of the DNNs left in the design space and trains them on the provided dataset. The performance results of this subset are then used to create a function that approximates the test-set accuracy for all DNNs in the space. Using this function, the design space is filtered further based on the required accuracy. If the optimization goal corresponding to one of the requirements above is provided, Shape_DNN proceeds to find the best shape in the design space according to the given goal. Pseudocode for the described process can be found in Algorithm 1.

4.1 Modeling Accuracy

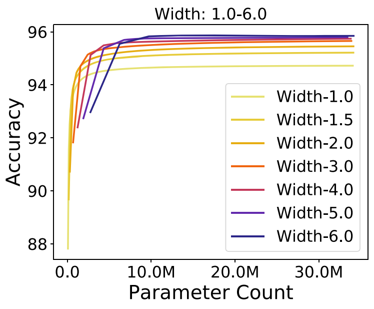
Filtering the design space based on the required size, latency, and energy consumption of a DNN is much less time-consuming when compared to the time spent on filtering by accuracy. This is because, to measure the latency or energy consumption of a DNN, it is sufficient to initialize it with random weights and use the techniques presented in the previous section. To measure the test-set accuracy of a given topology, on the other hand, it is required to train it first, which is a much longer process. Therefore, we reduce the time spent training the shapes in the design space by modeling their accuracy.

Despite the recent trend toward deeper DNN architectures, they come with a penalty of an increased network latency, as demonstrated earlier. The question of the importance of depth in neural networks was raised before [1] and it was empirically shown [26] that having multiple hidden layers with non-linearities is crucial to obtaining high accuracy on an image classification task. However, selecting the optimal value for depth is not trivial. Shape_DNN can help the user choose optimal values for depth and width.

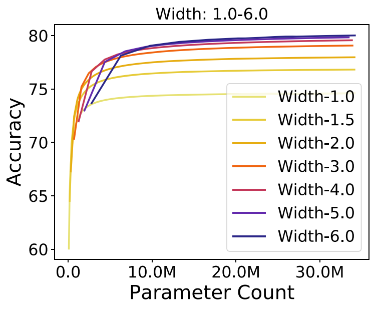
ALGORITHM 1: Tuning Width and Depth Using Shape_DNN

```
Input: Design space specification.
Input: Classification problem.
Input: $Constraints$.
Input: $OptimizationGoal$[optional].
Input: $ModelingStrategy$[optional].
Input: Training hyperparameters [optional].
Output: A set of shapes complying with $Constraints$ or the best shape out of this set with respect to
        $OptimizationGoal$ if one is provided.
begin
    $Shapes \longleftarrow InitializeDesignSpace()$
    if $MaxSize \in Constraints$ then
        $Shapes.RemoveLargerThan(MaxSize)$
    if $MaxLatency \in Constraints$ then
        $Shapes.BenchmarkLatencies()$
        $Shapes.RemoveSlowerThan(MaxLatency)$
    if $MaxEnergy \in Constraints$ then
        $Shapes.MeasureEnergyConsumption()$
        $Shapes.RemoveLessGreenThan(MaxEnergy)$
    $ChosenShapes \longleftarrow ModelingStrategy.Select(Shapes)$
    $RealAccuracies \longleftarrow ChosenShapes.Train()$
    $AccuracyModel \longleftarrow BuildModel(RealAccuracies)$
    $Shapes.PredictAccuracy(AccuracyModel)$
    if $MinAccuracy \in Constraints$ then
        $Shapes.RemoveLessAccurateThan(MinAccuracy)$
    if $no\ OptimizationGoal$ then
        return $Shapes$
    else
        $Shapes.OrderBy(OptimizationGoal)$
        for $BestShape \in Shapes$ do
            $Accuracy \longleftarrow BestShape.Train()$
            if $MinAccuracy \in Constraints\ and\ Accuracy < MinAccuracy$ then
                Accuracy below expected, continue with the next best shape.
            else
                Found the optimal shape.
                return $BestShape$
```

Although bigger ResNet architectures are more prone to overfitting the training set, the lowest achieved test-set error during training tends to be in inverse relationship with the number of parameters of a DNN [5]. To select the base features of our model, we created a dataset with the highest achieved percentages of test-set accuracy on the CIFAR-10 classification task for different shapes of ResNets. We split this dataset into training and test sets and experimented with the various base features for our model drawn out of a set shown in Equation (1), where d stands for the depth and w for the width. The base features shown in Equation (2) were selected because they had the lowest absolute values of s and t while the trained model produced a score of $R^2 > 0.96$ on the training and test sets. We define the function predicting the accuracy as in Equation (3) and



(a) CIFAR-10



(b) CIFAR-100

Fig. 6. Relationship between test-set accuracy and the number of parameters for CIFAR-10 and CIFAR-100. Different colors represent topologies of different width.

use linear regression to find the vector of coefficients v and bias b. The accuracy of this model and the choice of the trained shapes is analyzed in detail in Sections 4.2 and 4.3.

$$\{d^s w^t | s, t \in \mathbb{Z} | -5 \leq s, t \leq 0\} \tag{1}$$

$$x = \left(\frac{1}{w} \frac{1}{d} \frac{1}{w^2} \frac{1}{dw} \frac{1}{d^2} \frac{1}{w^3} \frac{1}{w^2 d} \frac{1}{wd^2} \frac{1}{d^3} \right) \tag{2}$$

$$y_p(x) = v^T x + b \tag{3}$$

Figure 6 shows the relation between the accuracy predicted by the fitted function and the parameter count for CIFAR-10 and CIFAR-100; each color line represents a topology of a different width. We present the results of achieved test-set accuracy alongside parameter count, since the number of parameters can be interpreted as a cost of increasing both width and depth. As can be observed from the chart, the choice of the best-performing topology in terms of accuracy depends on the number of parameters available for construction of a DNN. Since the number of parameters determines the size of the DNN, given a maximum size, it is possible to predict the choice of a best-performing topology.

4.2 Choosing DNN Shapes to Train

Modeling accuracy is an effective tool for reducing the time spent searching and navigating the design space of DNNs. It requires training a subset of DNNs in the design space; thus, it is important to train a sufficient number of DNNs and to choose the shapes to train such that the prediction accuracy of the fitted function is maximized while the training time is minimized.

We tested three strategies for choosing the shapes to train from the design space: random sampling, wider-shape sampling, and interval sampling from the list of shapes ordered lexicographically by width and depth. Wider-shape sampling means ordering samples by width and depth in ascending order and choosing the first m shapes. The advantage of this strategy over the others is that, on average, it results in less training time because small wider shapes are faster to train, and for every distinct width in the design space there is at least one sample trained provided that the number m of trainings performed is sufficiently large. Interval sampling showed the best results on all datasets in terms of prediction accuracy, followed by random sampling and wider-shape sampling. We attribute the superiority of interval sampling to the amount of noise present in the training data, that is, topologies of similar shape tend to perform relatively similarly, and stating the superiority of one shape over the other in terms of prediction accuracy with any sort of confidence requires a number of trainings of both to be performed with various initialization

Table 1. The Level of Noise (2 std. dev.) in MNIST, CIFAR-10, and CIFAR-100 and the Minimum Number of Shapes That is Required to Train for Design Spaces of Various Sizes So That the Mean Prediction Errors Fall Below the Amount of Noise for the Dataset

Dataset	Noise	Shapes Trained	Total Shapes	Mean Prediction Error
MNIST	0.1	16	16	N/A
		10	20	0.10
		20	40	0.09
		19	60	0.06
		26	80	0.09
CIFAR-10	0.4	17	17	N/A
		10	21	0.38
		21	43	0.20
		12	64	0.16
		17	86	0.13
CIFAR-100	0.6	8	17	0.42
		10	21	0.28
		13	42	0.35
		12	63	0.36
		17	85	0.38

The noise refers to the size of the probable range of values of prediction accuracy that a given DNN shape is likely to achieve when trained with various initialization parameters.

parameters. Since the training time of any given shape is high, training a single shape many times is infeasible for design space exploration. Interval sampling is advantageous because it covers the whole design space. This results in the effect of noise being less pronounced, which makes it possible to fit the function more accurately.

4.3 The Number of Trained DNNs

To judge the effectiveness of modeling test-set accuracy with any given number of training samples, we first calculated the level of noise in the training data. For each dataset, with the exception of ImageNet32×32, we selected multiple shapes and performed 10 trainings of each with different initialization parameters. The variance in the final prediction accuracy of these shapes on the test data was used as an indicator of noise, that is, tolerable level of error for our linear regression model. To simulate a task of modeling accuracy, we trained up to 86 different shapes from each dataset and created samples of various sizes from these shapes to mimic DNN design spaces. For each design space, we used various percentages of the total number of shapes as training data for our linear regression model. Comparing the mean prediction error of our model with the level of noise, we were able to choose the minimum number of shapes that needs to be trained to achieve a prediction accuracy within the tolerance level defined by the noise for each data set. The results can be found in Table 1. Notably, although the level of noise increases with the complexity of the classification task, the required minimum number of training changes only slightly. For example, the numbers for CIFAR-100 are close to those of CIFAR-10 in similar design spaces. For MNIST, we explored the design space with a lower step size between values of width because of the lower complexity of the classification problem. As can be observed, decreasing the step size requires training more samples, because the amount of noise in the training data also increases.

Training less shapes from each dataset would accelerate design space traversal at the expense of accuracy of the fitted function. Hence, to avoid compromising accuracy, we refrain from further reducing the number of trained DNNs. Table 1 also shows that, for larger design spaces, less training may be required to achieve acceptable prediction accuracy. For example, when increasing the size of the design space for CIFAR-10 from 43 to 64 and training less shapes – 12 instead of 21 – the linear regression model has lower prediction error, which might seem counter-intuitive. However, the important difference between these two cases is the choice of the trained shapes. Interval sampling diminishes the effect of noise in the training data for a larger design space, even though the number of trained shapes is low. We believe that there are several factors that need to be taken into account when selecting the appropriate number of shapes to train:

- complexity of the classification problem
- size of the design space
- step size

While the implementation of accurate decision making based on all of the aforementioned factors is out of the scope of this article, the user of Shape_DNN can choose among three modeling strategies: *thorough*, *standard*, and *optimistic*. The strategies are listed in descending order of the number of training samples. Depending on the selected strategy, the time spent traversing the design space can be reduced up to 3, 4, or 5 times in comparison to exhaustive grid search. For each strategy, we have defined a set of threshold values reflecting the size of the design space. Once the number of shapes in the design space crosses the threshold value, the percentage of shapes to be trained drops. To find suitable threshold values and percentages for each strategy, we first calculated the optimal number of shapes that is required to train for the three studied datasets with different levels of complexity of classification task and various step sizes in the design space, as shown in Table 1. Then, we roughly divided anticipated real-life design spaces into three categories of varying complexity of the classification task and expected step size: low-complexity tasks, which are similar to our simulations with MNIST, where a smaller step size is likely to be used; medium, covering the range of problems similar to CIFAR-10 and CIFAR-100; and high, intended for classification problems with complexity similar to ImageNet [13]. The standard strategy is the default and should be sufficient for most scenarios. The optimistic strategy can be advantageous for more complex classification tasks and large design spaces. In contrast, smaller design spaces with a small step size might demand the thorough modeling strategy to achieve higher prediction accuracy.

5 EVALUATION

We performed multiple experiments with different constraints and optimization goals. Since the classification of images in MNIST is relatively simple in comparison to CIFAR and ImageNet32×32, we focused on exploring the design space of thinner ResNet topologies for this dataset. In total, we performed experiments across five design spaces, which were initialized with the values provided in Table 2. The remaining hyperparameters followed the selection of Zagoruyko and Komodakis [28].

Because the narrow step size of the width in design space A, which we chose for MNIST, makes accuracy modeling more susceptible to noise, we followed our thorough modeling strategy for this dataset. For CIFAR-10 and CIFAR-100, we chose our standard modeling strategy. Lastly, for ImageNet32×32, we applied the optimistic modeling strategy because this classification task is more complex than the others. To test the robustness of Shape_DNN in scenarios where models overfit the training data, we also experimented with reducing the L2 regularization coefficient for ImageNet32×32. For the sake of presentation, we refer to this design space as D2, even though the shapes it contains are identical to D1. The results for MNIST, CIFAR-10, and CIFAR-100 are presented in Table 3, while the results for ImageNet32×32 can be found in Tables 4 and 5. Each

Table 2. The Shape Design Spaces for the ResNets Used in Our Experiments

		Depth			Width			
Design Space	Size	Min	Max	Step	Min	Max	Step	L2 regularization coefficient
A	135	10	60	6	1/16	1	1/16	0.0005
B	45	10	60	6	1	6	1	0.0005
C	180	10	60	6	1	6	1/4	0.0005
D1	119	16	53	6	2	21/2	1/2	0.0005
D2	119	16	53	6	2	21/2	1/2	0.0001

row shows the speedup achieved by our accuracy modeling strategy over the default exhaustive grid search as well as the percentage of shapes trained. Whereas the speedup indicates the effort relative to training the full set of shapes meeting the non-functional requirements, the percentage of shapes trained roughly indicates the effort relative to the other tasks in the same unfiltered design space.

As the last step of its workflow, Shape_DNN automatically trains the selected shape and verifies that its accuracy is above the required level if one is provided. If the prediction accuracy of the trained shape is not sufficient, Shape_DNN proceeds to train the next shape expected to perform above the required accuracy level, following the order defined by the optimization goal. In our experiments with MNIST, CIFAR-10, and CIFAR-100, the accuracy of the initial selection was sufficiently high in the majority of cases. In the remaining cases, the second choice always achieved the necessary level of accuracy. The overhead of training the underperforming shape was taken into account when calculating the speedup in Table 3. Because training networks for ImageNet32×32 is way more expensive than for the others, limited access to computational resources did not allow us to execute our workflow in its entirety for this dataset. For both design spaces D1 and D2, we were able to train all the initially chosen shapes and show the actual accuracy next to the prediction (Tables 4 and 5). In the minority of cases, the actual accuracy is below the required one. Those are highlighted. If this happens, Shape_DNN normally continues searching the design space for the network shape complying with the accuracy requirements. However, lacking time to compute, this part of the design space exploration was not performed for ImageNet32×32. Consequently, the speedup shown in Tables 4 and 5 does not include the time required to perform additional training.

In our analysis of CIFAR-10, we compare our results to depth-scaled ResNets of width 1 as baseline, equivalent to those introduced by He et al. [10]. While the concentration on this dataset as a showcase was economically motivated, we believe that it already demonstrates the significant optimization potential of tuning ResNets with Shape_DNN.

5.1 MNIST

To explore the trade-off between DNN model size and its accuracy, we selected several constraints on the maximum DNN size and for each size we used Shape_DNN to find the most accurate shape in design space A. We also put an upper bound on the maximum amount of energy consumption to mimic the scenario of deployment on a battery-powered device. A larger allowed maximum DNN size results in less shapes being eliminated based on their size. However, the bigger number of remaining shapes reduces the fraction of shapes that need to be trained among them, leading to greater speedup achieved via accuracy modeling vs. exhaustive grid search. In spite of a tight size threshold of 20KB, Shape_DNN was still able to identify a ResNet architecture that achieved prediction accuracy of 99%, consuming just 19KB of memory and carrying 4668 parameters.

Table 3. Results of Shape_DNN Applied to MNIST, CIFAR-10, and CIFAR-100

				Constraints				Results					
Dataset	Space	Goal	Batch size	Size (MB)	Latency (ms)	Energy (kJ)	Acc. (%)	Shape (depth, width)	Acc. (%)	Latency (ms)	Energy (kJ)	Speedup	Trained (%)
MNIST	A	Accuracy	1024	≤0.02	-	≤3	-	(22, 2/16)	99.00	-	-	1x	10
				≤0.04	-	≤3	-	(40, 2/16)	99.26	-	-	1x	14
				≤0.06	-	≤3	-	(58, 2/16)	99.25	-	-	2x	10
				≤0.10	-	≤3	-	(52, 2/16)	99.51	-	-	2x	12
				≤0.16	-	≤3	-	(28, 5/16)	99.54	-	-	2x	17
				≤0.40	-	≤3	-	(34, 7/16)	99.60	-	-	3x	17
				≤1.00	-	≤6	-	(28, 13/16)	99.67	-	-	3x	24
	A	Energy	1024	≤2	-	≤3	≥95.0	(10, 2/16)	97.28	-	304.04	3x	33
				≤2	-	≤3	≥98.0	(10, 3/16)	98.49	-	378.77	3x	33
				≤2	-	≤3	≥99.0	(10, 5/16)	99.09	-	519.00	3x	33
				≤2	-	≤3	≥99.2	(16, 3/16)	99.21	-	608.52	3x	33
				≤2	-	≤3	≥99.4	(16, 5/16)	99.42	-	900.75	3x	33
				≤2	-	≤3	≥99.6	(34, 7/16)	99.60	-	2921.41	3x	33
CIFAR-10	B	Accuracy	1	≤2	≤4	-	-	(34, 1)	93.41	-	-	1x	13
				≤4	≤4	-	-	(16, 2)	94.15	-	-	1x	20
				≤8	≤4	-	-	(16, 3)	94.75	-	-	1x	31
				≤16	≤4	-	-	(28, 3)	95.55	-	-	1x	40
				≤32	≤4	-	-	(22, 5)	95.88	-	-	2x	29
				≤64	≤4	-	-	(40, 5)	95.74	-	-	2x	33
	B	Accuracy	128	-	≤4	-	-	(22, 1)	92.55	-	-	1x	9
				-	≤6	-	-	(16, 2)	94.15	-	-	1x	17
				-	≤8	-	-	(16, 3)	94.75	-	-	1x	27
				-	≤12	-	-	(22, 3)	95.30	-	-	2x	27
				-	≤24	-	-	(28, 4)	95.75	-	-	2x	36
	C	Latency	128	≤128	≤24	-	≥94.0	(22, 7/4)	94.32	5.3	-	4x	25
				≤128	≤24	-	≥94.5	(16, 11/4)	94.67	6.8	-	4x	25
				≤128	≤24	-	≥95.0	(16, 15/4)	95.39	9.6	-	4x	25
				≤128	≤24	-	≥95.5	(22, 16/4)	95.80	14.8	-	4x	25
CIFAR-100	B	Accuracy	1	≤2	≤4	-	-	(34, 1)	70.93	-	-	1x	13
				≤4	≤4	-	-	(16, 2)	72.76	-	-	1x	20
				≤8	≤4	-	-	(34, 2)	75.36	-	-	1x	23
				≤16	≤4	-	-	(28, 3)	77.51	-	-	1x	42
				≤32	≤4	-	-	(34, 4)	79.02	-	-	2x	29
				≤64	≤4	-	-	(28, 5)	79.83	-	-	2x	36
	B	Latency	128	-	-	-	≥77.5	(16, 5)	77.86	15.01	-	3x	33
				-	-	-	≥78.0	(28, 4)	78.70	21.21	-	3x	33
				-	-	-	≥78.5	(28, 4)	78.70	21.21	-	3x	33
				-	-	-	≥79.0	(22, 5)	79.46	23.20	-	3x	33
				-	-	-	≥79.5	(28, 5)	79.83	30.87	-	3x	33
				-	-	-	≥80.0	(46, 5)	80.20	54.43	-	3x	33

Design spaces A, B and C are defined in Table 2. The batch size refers to the one used to measure latency and energy rather than the one used to train the neural network.

Table 4. Results of Shape_DNN Applied to ImageNet32×32 in Design Space D1, as Defined in Table 2

				Constraints	Results							
Data Set	Space	Goal	Batch size	Accuracy (%)	Shape (depth, width)	Acc. Ex-pected (%)	Acc. Real (%)	Latency (ms)	Energy (kJ)	Size (MB)	Speedup	Trained (%)
ImageNet32x32	D1	Latency	1	≥55.5	(22, 6.5)	55.95	55.83	6.11	-	-	5x	20
				≥56.0	(28, 5.5)	56.02	55.94	6.44	-	-	5x	20
				≥56.5	(22, 7.0)	56.58	56.75	6.51	-	-	5x	20
				≥57.0	(16, 10.0)	57.05	56.63	7.27	-	-	5x	20
				≥57.5	(34, 6.0)	57.74	57.69	8.56	-	-	5x	20
				≥58.0	(22, 8.5)	58.06	57.67	9.36	-	-	5x	20
				≥58.5	(22, 9.5)	58.81	58.33	10.26	-	-	5x	20
				≥59.0	(22, 10.0)	59.13	58.76	10.72	-	-	5x	20
				≥59.5	(28, 9.5)	59.85	59.75	13.50	-	-	5x	20
				≥60.0	(28, 10.0)	60.13	59.92	14.28	-	-	5x	20
				≥60.5	(34, 9.5)	60.52	60.52	16.92	-	-	5x	20
				≥61.0	(40, 10.0)	61.23	61.23	21.33	-	-	5x	20
				≥61.5	(46, 9.5)	61.74	61.74	23.59	-	-	5x	20
		Energy	1	≥55.5	(16, 8.5)	55.76	55.64	-	671.21	-	5x	20
				≥56.0	(28, 5.5)	56.02	55.94	-	685.58	-	5x	20
				≥56.5	(22, 7.0)	56.58	56.75	-	743.67	-	5x	20
				≥57.0	(34, 5.5)	57.09	57.16	-	850.61	-	5x	20
				≥57.5	(22, 8.0)	57.64	57.64	-	904.81	-	5x	20
				≥58.0	(22, 8.5)	58.06	57.67	-	1003.01	-	5x	20
				≥58.5	(22, 9.5)	58.81	58.33	-	1141.33	-	5x	20
				≥59.0	(46, 6.0)	59.03	58.94	-	1313.57	-	5x	20
				≥59.5	(34, 8.0)	59.60	59.21	-	1502.64	-	5x	20
				≥60.0	(34, 9.0)	60.24	59.93	-	1761.47	-	5x	20
				≥60.5	(34, 9.5)	60.52	60.52	-	1900.55	-	5x	20
				≥61.0	(46, 9.0)	61.14	61.10	-	2462.69	-	5x	20
				≥61.5	(46, 9.5)	61.74	61.74	-	2656.40	-	5x	20
		Size	1	≥55.5	(52, 3.5)	55.54	55.95	-	-	38.0	5x	20
				≥56.0	(46, 4.0)	56.07	56.20	-	-	43.2	5x	20
				≥56.5	(52, 4.0)	56.69	56.98	-	-	49.4	5x	20
				≥57.0	(46, 4.5)	57.23	57.23	-	-	54.5	5x	20
				≥57.5	(52, 4.5)	57.59	57.67	-	-	62.3	5x	20
				≥58.0	(52, 5.0)	58.33	58.46	-	-	76.8	5x	20
				≥58.5	(34, 7.0)	58.79	58.52	-	-	92.8	5x	20
				≥59.0	(46, 6.0)	59.03	59.11	-	-	96.3	5x	20
				≥59.5	(46, 6.5)	59.50	59.16	-	-	112.9	5x	20
				≥60.0	(40, 8.0)	60.17	60.03	-	-	145.6	5x	20
				≥60.5	(40, 8.5)	60.55	60.55	-	-	164.3	5x	20
				≥61.0	(46, 9.0)	61.14	61.10	-	-	215.4	5x	20
				≥61.5	(46, 9.5)	61.74	61.74	-	-	239.8	5x	20

The batch size refers to the one used to measure latency and energy rather than the one used to train the neural network. Real accuracies that are lower than required are highlighted with gray background.

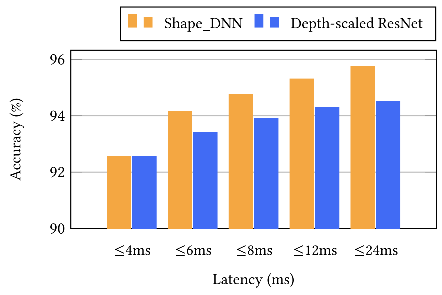
Table 5. Results of Shape_DNN Applied to ImageNet32×32 in Design Space D2, as Defined in Table 2

				Constraints	Results							
Data Set	Space	Goal	Batch size	Accuracy (%)	Shape (dep., wid.)	Acc. Expected (%)	Acc. Real (%)	Latency (ms)	Energy (kJ)	Size (MB)	Speedup	Trained (%)
ImageNet32x32	D2	Latency	1	≥50.0	(16, 5.5)	50.49	50.54	4.26	-	-	5x	20
				≥50.5	(16, 6.0)	51.23	51.24	4.60	-	-	5x	20
				≥51.0	(16, 6.0)	51.23	51.24	4.60	-	-	5x	20
				≥51.5	(16, 7.0)	52.61	52.61	5.39	-	-	5x	20
				≥52.0	(16, 7.0)	52.61	52.61	5.39	-	-	5x	20
				≥52.5	(16, 7.0)	52.61	52.61	5.39	-	-	5x	20
				≥53.0	(16, 8.0)	53.33	53.56	6.27	-	-	5x	20
				≥53.5	(16, 8.5)	53.71	53.65	6.76	-	-	5x	20
				≥54.0	(16, 9.0)	54.05	53.93	7.09	-	-	5x	20
				≥54.5	(16, 10.0)	54.64	54.56	7.67	-	-	5x	20
				≥55.0	(22, 8.0)	55.07	55.07	9.40	-	-	5x	20
		Energy	1	≥50.0	(22, 4.0)	50.10	50.04	-	353.25	-	5x	20
				≥50.5	(16, 6.0)	51.23	51.24	-	390.44	-	5x	20
				≥51.0	(16, 6.0)	51.23	51.24	-	390.44	-	5x	20
				≥51.5	(28, 4.0)	51.77	51.77	-	461.03	-	5x	20
				≥52.0	(22, 5.0)	52.05	52.31	-	464.07	-	5x	20
				≥52.5	(16, 7.0)	52.61	52.61	-	500.94	-	5x	20
				≥53.0	(22, 6.0)	53.38	53.63	-	574.86	-	5x	20
				≥53.5	(34, 4.5)	53.55	53.55	-	669.19	-	5x	20
				≥54.0	(28, 5.5)	54.01	53.95	-	685.58	-	5x	20
				≥54.5	(28, 6.0)	54.54	54.50	-	759.92	-	5x	20
				≥55.0	(22, 8.0)	55.07	55.07	-	904.81	-	5x	20
		Size	1	≥50.0	(34, 3.0)	50.07	49.85	-	-	17.5	5x	20
				≥50.5	(40, 3.0)	50.96	50.92	-	-	21.0	5x	20
				≥51.0	(22, 4.5)	51.18	51.30	-	-	23.0	5x	20
				≥51.5	(34, 3.5)	51.56	51.66	-	-	23.7	5x	20
				≥52.0	(52, 3.0)	52.10	52.10	-	-	28.0	5x	20
				≥52.5	(34, 4.0)	52.68	52.68	-	-	30.8	5x	20
				≥53.0	(40, 4.0)	53.43	53.47	-	-	37.0	5x	20
				≥53.5	(34, 4.5)	53.55	53.81	-	-	38.8	5x	20
				≥54.0	(28, 5.5)	54.01	53.95	-	-	45.9	5x	20
				≥54.5	(28, 6.0)	54.54	54.50	-	-	54.4	5x	20
				≥55.0	(46, 4.5)	55.22	55.22	-	-	54.5	5x	20

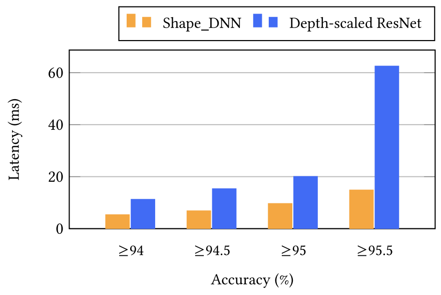
The batch size refers to the one used to measure latency and energy rather than the one used to train the neural network. Real accuracies that are lower than required are highlighted with gray background.

Surprisingly, the size of the smallest model is 50 times less than that of the biggest one while the difference in test-set accuracy is less than 0.7%. This type of design space exploration can be useful for deployment on small devices with limited memory and for various applications with different levels of error tolerance.

Another useful application is the minimization of energy consumption while preserving a desired level of prediction accuracy. We defined several constraints on the minimum DNN accuracy and used Shape_DNN to find the most energy-efficient shapes in design space A for processing a



(a) Maximizing accuracy under a constraint on latency.



(b) Minimizing latency under a constraint on accuracy.

Fig. 7. Comparison between shapes produced by Shape_DNN in two different experiments and default depth-scaled ResNets. Latencies are measured for a batch size of 128.

batch of 1024 images. The requirement on minimum accuracy is a functional requirement, as opposed to size, as in the previous scenario, which means that none of the shapes was filtered based on non-functional requirements and all 135 are considered for the accuracy modeling phase; thus, the speedup achieved in each experiment is the same. It can be observed that higher accuracy requirements drastically increase energy consumption. For example, the shape (16, 3/16) is nearly five times more energy efficient than (34, 7/16), while decreasing the test-set accuracy only by 0.39%.

5.2 CIFAR-10

For CIFAR-10, we performed similar tests with the goal of optimizing accuracy under various constraints on the size of a DNN and the same maximum latency of 4ms for a single input image. This kind of constraint can be advantageous for real-time applications; another benefit of a small latency is the reduced training time, which means that the time spent on searching the design space is also reduced. The attentive reader might notice that the prediction accuracy of the produced DNN degrades when increasing the maximum DNN size from 32MB to 64MB. For example, shape (22, 5) is more accurate than its deeper alternative (40, 5) while being smaller in size; however, Shape_DNN chooses the latter. This is because only a part of the shapes in the design space is trained and our model wrongly predicted (40, 5) to be more accurate than (22, 5). Nevertheless, the architecture chosen by Shape_DNN is still near optimal.

Moreover, we maximized test-set accuracy under various latency constraints on a batch of 128 images in design space B. Figure 7(a) compares the results obtained by Shape_DNN with depth-scaled shapes of width 1, which were selected manually from a set of 544-layer-deep, pretrained shapes to satisfy the latency requirement. Obviously, tuned shapes substantially outperform their deeper counterparts with the default width of 1 in the majority of cases. The advantage of wider shapes in these types of scenarios can be partly attributed to their ability to hold more parameters than their thinner alternatives without exceeding a given latency threshold.

To further illustrate the advantage of using Shape_DNN, we selected several values for the minimum test-set accuracy and searched for the fastest shapes fulfilling those requirements in design space C. We compared the latencies of these shapes on a batch of 128 images with the fastest of the default depth-scaled implementations of ResNets complying with the same requirements. The results are shown in Figure 7(b). As can be observed, DNNs produced by Shape_DNN are considerably faster than the default depth-scaled implementations. For example, shape (22, 4), which satisfies the minimum test-set accuracy of 95.5%, is 4.22 times faster than the default depth-scaled shape (364, 1) on a batch of 128 images.

5.3 CIFAR-100

As with the other datasets, we used Shape_DNN to find the most accurate topologies under different constraints on the model size in design space B. CIFAR-100 is a considerably harder classification task compared to CIFAR-10 and MNIST, with the number of output classes being 10 times higher. Therefore, increasing the size of a DNN results in a bigger improvement of its prediction accuracy in comparison to CIFAR-10 and MNIST.

We also performed several experiments with various constraints on test-set accuracy and the goal of minimizing latency on a batch of 128 images in the same design space. As is evident from the results, wider shapes succeed better in reconciling accuracy with speed.

5.4 ImageNet32×32

For our experiments with ImageNet32×32, we set the learning rate to 0.01, as suggested by Chrabaszcz et al. [5]. Input normalization was also performed in the same manner. However, different from Chrabaszcz et al., who dropped the learning rate in epochs 10, 20, and 30, we reduced it to a fifth of its value upon reaching epochs 20, 40, and 60 because, as opposed to them, we did not double the size of the training data using horizontal image flips but rather incorporated that preprocessing step into our data augmentation procedure during training. We trained the shapes in design spaces D1 and D2 for 60 and 40 epochs, respectively. All other hyperparameters equaled those used in our experiments with CIFAR-10 and CIFAR-100. Because of varying resource availability, the latencies reported for design space D1 were obtained on a Tesla K20X GPU. All other experiments were performed on a Tesla K20 GPU.

In all of our experiments with ImageNet32×32, we constrain accuracy only while trying to minimize either latency, energy consumption, or the size of the DNNs. Thanks to the optimistic modeling strategy employed in all of our experiments, we had to train only 20% of all the shapes. As can be observed in Tables 4 and 5, in the majority of cases the first shape chosen by Shape_DNN satisfies the accuracy requirement. Those cases where the real accuracy is below the required level are highlighted with a gray background. However, even in these cases, the real accuracy is remarkably close to the expected one. We attribute the majority of mispredictions to the noise in the dataset, because in many of the experiments, in order to maximize the optimization objective, Shape_DNN chooses the shape that is expected to perform just above the required level of accuracy.

When optimizing latency in design space D1, the chosen shapes are deeper for higher values of the minimum accuracy. In contrast, with the same optimization goal in design space D2, Shape_DNN tends to choose wider shapes as the required minimum accuracy increases. The reason is that the accuracy requirements for our experiments in design space D2 are lower than in D1, and the accuracy model of Shape_DNN expects even the shallowest shapes to satisfy the constraints.

As the constraint on the minimum test-set accuracy increases so does the energy consumption of complying shapes, for example, the last 1% improvement in accuracy costs 40% more energy in design space D1 and 32% more in design space D2. For experiments with the same value of minimum accuracy, the smallest, most energy-efficient and fastest shapes are usually different. Therefore, choosing the DNN shape randomly or testing only a small amount of different shapes without modeling their accuracy is likely to yield suboptimal architectures.

6 CONCLUSION AND OUTLOOK

In this article, we presented our approach to navigating the design space of various ResNet topologies defined by the two parameters of width and depth. We have shown how these parameters affect the size, energy consumption, latency, and accuracy of the resulting DNN. Based on our observations, we have constructed a set of tools that can be used to efficiently search the design space of DNN and meet functional as well as non-functional requirements, including size, speed, and energy consumption.

There are several shortcomings of our approach, which we will address in the future. First, the DNN training time is by far its most time-consuming part. We believe it can be further reduced using various methods in addition to training several shapes in parallel on a multi-server cluster. For example, by modeling the learning progress, it could be possible to abruptly terminate the training of underperforming topologies so that time is not wasted on the whole training process. Instead of training each shape from scratch, knowledge transfer techniques can be used to accelerate design-space exploration, and the like. In some scenarios, it might be necessary to produce a DNN that is optimal with respect to more than one property; for example, both size and energy might be equally important when running on a mobile device. For these scenarios, Shape_DNN could be easily extended to support a custom scoring function, so that a combination of several properties is considered for tuning. Also, we plan to integrate recently developed DNN architectures such as DenseNet and NASNet into Shape_DNN. We believe that support for the aforementioned features will make Shape_DNN more useful for real-life applications.

ACKNOWLEDGMENTS

This work has been supported by the Excellence Initiative of the German Federal and State Governments and the Graduate School of Computational Engineering at TU Darmstadt and by the Hessian LOEWE initiative within the Software-Factory 4.0 project. Some of the calculations for this research were conducted on the Lichtenberg Cluster of TU Darmstadt, others on the cluster of the graduate school. The authors would especially like to thank the IT administrators of the graduate school and specifically Christian Schmitt for helping set up the computing resources.

REFERENCES

[1] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2654–2662.

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing neural network architectures using reinforcement learning. (2016). arxiv:1611.02167

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

[4] Martin Burtscher, Ivan Zecena, and Ziliang Zong. 2014. Measuring GPU power with the K20 built-in sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs*. ACM, 28.

[5] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2017. A downsampled variant of ImageNet as an alternative to the CIFAR datasets. (2017). arxiv:1707.08819

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 3123–3131.

[7] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. (2015). arxiv:1510.00149

[8] Sayed Hadi Hashemi, Shadi A. Noghabi, and William Gropp. 2016. Performance modeling of distributed deep neural networks. (2016). arxiv:1612.00521

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision*. Springer, 630–645.

[11] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2261–2269.

[12] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. (2016), 1–13. arxiv:1602.07360

[13] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 248–255.

[14] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. *Science Department, University of Toronto, Tech.* (2009), 1–60.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Vol. 1 (NIPS'12)*. Curran Associates Inc., 1097–1105.

[16] Jens Lang and Gudula Rünger. 2013. High-resolution power profiling of GPU functions using low-resolution measurement. In *Proceedings of the European Conference on Parallel Processing*, Felix Wolf, Bernd Mohr, and Dieter an Mey (Eds.). Springer, 801–812.

[17] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. 2007. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*. 473–480.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, Vol. 86. 2278–2323.

[19] Yann LeCun and Corinna Cortes. [n.d.]. MNIST handwritten digit database. Retrieved November 21, 2018 from http://yann.lecun.com/exdb/mnist/.

[20] Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. 2017. Evolving deep neural networks. (2017). arxiv:1703.00548

[21] Renato Negrinho and Geoff Gordon. 2017. Deeparchitect: Automatically designing and training deep architectures. (2017). arxiv:1704.08792

[22] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. 2017. Paleo: A performance model for deep neural networks. In *Proceedings of the International Conference on Learning Representations*.

[23] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: ImageNet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*. Springer, 525–542.

[24] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. (2014). arxiv:1412.6550

[25] Pytorch Core Team. 2017. Pytorch: Tensors and dynamic neural networks in python with strong GPU acceleration.

[26] Gregor Urban, Krzysztof J. Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. 2016. Do deep convolutional nets really need to be deep and convolutional? (2016). arxiv:1603.05691

[27] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. (2017). arxiv:1611.05128

[28] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. (2016). arxiv:1605.07146

[29] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. (2016). arxiv:1606.06160

[30] Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. (2016). arxiv:1611.01578

[31] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2017. Learning transferable architectures for scalable image recognition. (2017). arxiv:1707.07012

Received May 2018; revised September 2018; accepted October 2018