Data-Driven Compression of Convolutional Neural Networks

RAMIT PAHWA, Adobe
MANOJ GHUHAN ARIVAZHAGAN, Adobe Research
ANKUR GARG, The University of Texas at Austin, USA
SIDDARTH KRISHNAMOORTHY, Indian Institute of Technology, Kanpur, India
ROHIT SAXENA, Indian Institute of Technology, Roorkee, India
SUNAV CHOUDHARY, Adobe Research

Deploying trained convolutional neural networks (CNNs) to mobile devices is a challenging task because of the simultaneous requirements of the deployed model to be fast, lightweight and accurate. Designing and training a CNN architecture that does well on all three metrics is highly non-trivial and can be very time-consuming if done by hand. One way to solve this problem is to compress the trained CNN models before deploying to mobile devices. This work asks and answers three questions on compressing CNN models automatically: a) How to control the trade-off between speed, memory and accuracy during model compression? b) In practice, a deployed model may not see all classes and/or may not need to produce all class labels. Can this fact be used to improve the trade-off? c) How to scale the compression algorithm to execute within a reasonable amount of time for many deployments? The paper demonstrates that a model compression algorithm utilizing reinforcement learning with architecture search and knowledge distillation can answer these questions in the affirmative. Experimental results are provided for current state-of-the-art CNN model families for image feature extraction like VGG and ResNet with CIFAR datasets.

Additional Key Words and Phrases: model compression, reinforcement learning, transfer learning

Manuscript submitted to ACM

ACM Reference format:
Ramit Pahwa, Manoj Ghuhan Arivazhagan, Ankur Garg, Siddarth Krishnamoorthy, Rohit Saxena, and Sunav Choudhary. 2019. Data-Driven Compression of Convolutional Neural Networks. 1, 1, Article 1 (December 2019), 17 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Convolutional Neural Networks (CNNs) have enabled monumental progress in many computer vision tasks over the past five years, achieving and even surpassing human level cognition [16, 22, 27]. Such networks are often carefully designed and have become very deep and large, especially the ones that claim to achieve state-of-the-art results. For example, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) for the year 2015 was won by a deep neural network from the ResNet [12] family with 152 layers and obtained a 3.57% error rate. Unfortunately, very deep networks can have a large memory footprint and become slow during inference. While this may not be a big problem when the trained model is deployed as a cloud service, it becomes a deal breaker for deployments on to the vast majority of comparatively resource constrained mobile devices and embedded systems.

Compressing trained CNNs has been suggested as one avenue to tame very large and deep networks for the purpose of deployment to resource constrained devices. While there has been significant progress in this direction recently (see the survey in [7]), it remains an active area of research with several open questions and challenges. One observation made in [7] is that the various approaches to compressing CNNs are somewhat orthogonal which raises the question "What are the right principles to employ for compressing families of CNN model architectures?" Yet another question is whether smaller/shallower neural networks with faster inference can come (robustly) close to the state-of-the-art results achieved by the deep CNN families, and if this could be done without the trial and error overhead of hand designed model architectures. In this work, we pose and study three related questions:

(1) For deploying models to resource constrained devices, the accuracy, memory footprint and inference speed of the deployed model are important considerations. How do we control the trade-off between these quantities (henceforth, referred to as the 'AMS trade-off')?
(2) In most applications, a deployed model will only see a reduced diversity of input data and/or will need to produce only a subset of the possible class labels. Can this aspect be used to improve the AMS trade-off? As a crude example, consider a neural network deployed on a self-driving car for identifying objects in its path. Such a network might have been trained on

the ImageNet dataset, but it only needs to be able to classify objects into two categories, viz. objects for which it needs to slow down or stop, and everything else.

(3) Current algorithms for model compression are fairly compute intensive. Depending on the particular algorithm and the compression targets that need to be achieved, it could take anywhere from a few hours to a couple of days to get a satisfactory result. If several different compressed models need to be generated, is it possible to reduce the amortized execution time? As a real-world example, consider a deep neural network that needs to power the user experience on a mobile application. Since users have different behavioral patterns, there is a case for deploying compressed models that are personalized to users/user segments. Scaling such a 'personalized compression' task beyond a few users becomes prohibitive very quickly.

Our contributions are to help answer the above questions and we do so by developing an algorithm for compressing CNNs. We draw upon prior work in [2] to utilize reinforcement learning (RL) to learn a compression policy for a given CNN trained on a given dataset, where compression is via architecture search and the reward function includes compression rate and accuracy of a candidate architecture (student model) trained using knowledge distillation [14, 24] from the given CNN (teacher model). We name our method as Data-driven Compression (DDC) and add the following over and above the contributions in [2].

(1) Besides compression rate and accuracy, we include inference time as a component of the reward function for RL. Further, we design reward functions with user definable performance thresholds for each of the three metrics in the AMS trade-off. The thresholds allow for control of the operating point in the AMS trade-off space for different deployment targets.

(2) We demonstrate that the compression policy learned on the full dataset generalizes to the compression task *w.r.t.* data subsets having fewer class labels. In other words, using the compression policy learned on the full dataset gives a *better* AMS trade-off on the compression task *w.r.t.* a data subset than executing the compression from scratch with a policy learned on the data subset. Moreover, using the former compression policy leads to much faster completion (5x faster) of the compression task *w.r.t.* data subsets than compressing from scratch. This leads to much smaller amortized execution time as compared to other model compression algorithms when multiple compressed models need to be generated while maintaining an impressive AMS trade-off for each compressed model. We conduct extensive experiments with ResNet and VGGnet model families trained on CIFAR datasets to demonstrate these results. Finally, we note that the compression policy transfer claims and supporting experiments in [2] are of an orthogonal nature. Therein, it is claimed that the learned compression policy generalizes to the compression task on other models from the same architecture family, *e.g.* compression policy learned on ResNet-18 can generalize to compress ResNet-34 models.

2 RELATED WORK

Reducing the depth of the networks [17] and utilizing less expensive operations, such as depthwise convolutions [15] and group convolutions [7] had gained momentum primarily due to their compactness and ease of deployment to restricted environment. These structures are special and hand designed. These networks have been almost entirely superseded by Architecture Search [18, 32] and Compression [13, 28]. In this section, we discuss the previous approaches that address the problem of compressing a given network architecture to smaller networks. There are mainly three such approaches, *viz.* pruning, knowledge distillation, and architecture search.

The pruning approach [1, 9, 20, 26] removes the neural network weights that contribute very little towards the performance of the model. A known issue with pruning is that it can over compress and damage the network beyond repair [21]. Further, there are very few human controls in the pruning method. In other words, metrics of interest like inference latency, accuracy and compression ratio cannot be directly optimized.

The knowledge distillation approach [3, 14, 24] trains a smaller network architecture (student) by utilizing the outputs of the original network (teacher). However, this approach is limited by the need to devise the student architecture.

Given a neural network, the architecture search approach involves searching for a smaller architecture (student) in the teacher architecture space that can display performance close to the original neural network. In general, brute force search through smaller architectures is computationally expensive. Recently, more principled search methods based on RL have been proposed [4, 31]. Furthermore, design of structured search spaces for good architectures has been undertaken using RL [32] and using evolutionary algorithms [22, 23]. More recently, Bayesian optimization has been proposed for hyper-parameter tuning [18]. This system (called Auto-Keras) also searches architecture from scratch. Searching an architecture from scratch [28, 32] has its limitation as it takes unrealistic time to search an optimal architecture for large datasets. These methods are limited when considering metrics that need to be controlled when deploying to resource constrained devices. Some of these restrictions have been recently incorporated in the architecture search space design [2, 6, 8, 28] to control the trade-off between performance and architectural complexity. For example, [2, 5] reduce the search complexity by restricts the student architecture search space to that of the original model's architecture. [5] uses Bayesian optimization to compress the original teacher model to new student architecture instead of searching from scratch. We build on this premise restricting our search space to that of the teacher model. This is computationally less expensive than searching architectures from scratch. We also introduce threshold-ed reward to further enforce search of optimal architecture which resides within the resource restrictions. A

Table 1. Feature Comparison of RL based methods for architecture search. NAS: Neural Architecture Search [32], AMC: AutoML for Model Compression [13], N2N: Network to Network Compression [2], MnasNet: Platform-Aware Neural Architecture Search [28], DDC: Data Driven Compression

	NAS	AMC	N2N	MnasNet	DDC
Accuracy Optimization	✓	✓	✓	✓	✓
Memory Optimization			✓		✓
Latency Optimization		✓		✓	✓
Threshold based Search				✓	✓
RNN Policy network	✓		✓		✓
Fast exploration on GPU		✓	✓	✓	✓
Distillation from Teacher			✓		✓

comparison of the features of different RL based methods for architecture search is provided in Table 1.

3 APPROACH

Our approach closely follows network to network compression [N2N] work done in [2], which introduces compressing down from high performing teacher models by modelling it a Markov Decision Process (MDP). Empirically by visualizing the activation's of teacher model, we observe the presence of redundant filter in each layer. We aim to automatically find the redundant layers in the network. To this extent, we train an RL agent to predict binary actions to keep or remove a layer, then update our agent by encouraging exploration of smaller, faster and more accurate model using thresholded reward functions. We systematically reduce the teacher model by sequentially deciding whether to keep an entire layer in the network architecture of the teacher model. This process is in contrast to [13] which utilizes actor-critic network to decide on the fraction of filters to keep in particular. Formally, our state space $\mathbb{S}$ comprises of the all the architectures obtained by compressing the teacher model, *i.e.* by removing layers from the teacher's architecture. The action set $\mathbb{A}$ consists of binary decision variable $a_t \in \{0, 1\}$, which enables us to control which layer to remove/keep in the teacher network.

3.1 Problem Definition

Compression of deep neural network can be achieved by parameter reduction. Parameter reduction can be achieved in two ways: First, is pruning individual unimportant elements in the weight matrix [11] which achieves high degree of compression while preserving accuracy. The only drawback of such algorithms is that it requires specialized hardware such as EIE [10] as the resultant weight matrix is sparse and irregular. On the other hand, structured pruning aims to remove entire regions (*e.g.* channels, layers, block, *etc.*). The resultant weight matrix is regular and can be accelerated directly with existing hardware and libraries. Our goal is reducing our complex teacher architecture, by finding out the irrelevant layer/layers which contribute little to the overall performance of the network. Let us consider VGG network [25], this network is a deep convolutional neural network. (VGG network is sequential in nature and typically have varying depth between 11 to 19). Now our aim is to remove a convolution layer/layers from the above architecture without perturbing its performance.

3.2 Recurrent Policy Network

We leverage RL to search for an optimal architecture. Unlike [28, 30, 32] which searches the optimal architecture from scratch.We restrict our state space to just the Teacher model Architectures (*viz.* VGG and Resnet) leading exponential reduction in architecture search time. We compress deep neural network by removing redundant layers from the network following which we use heuristic based reduction of filters [21] to further decrease the footprint of the model, resulting in a compact student model which has comparable performance to its respective teacher and is more computationally efficient. Each layer in the teacher's architecture is characterized by the following tuple:

$$L_t = (t, k, s, p, n, s_{start}, s_{end}) \tag{1}$$

where t represents the current layer under consideration, k is kernel size, s is stride and n is the number of outputs (filters). To cater for the presence of residual connections in Resnet [12] architecture family, we use two additional parameters s_{start} and s_{end} to explicitly inform the policy network of a skip connection. This allows us to distinctly represent the layers of the network under consideration. We feed the layer representation L_t to a LSTM unit which outputs a hidden network embedding h_t. Here, the action to remove the current layer L_t affects the preceding L_{t-1} and succeeding L_{t+1} layers thus, we use a Bi-directional LSTM network which is illustrated in **figure-1**. The binary action a_t which determines the presence or absence of the current layer depends upon the hidden state from both forward h_t and backward direction h_t as well as the current layer representation L_t. Formally, the policy is defined as:

$$\pi(a_t | h_{t-1}, h_{t+1}, x_t) \tag{2}$$

where π represents the policy. The produced network architecture needs to satisfy the user specific device restriction, to model this we propose a *threshold-ed reward function* to specify bounds on the accuracy a, size c and inference latency l. This enables us to search for efficient network architectures in a systematic manner. The architectures searched satisfy accuracy requirements

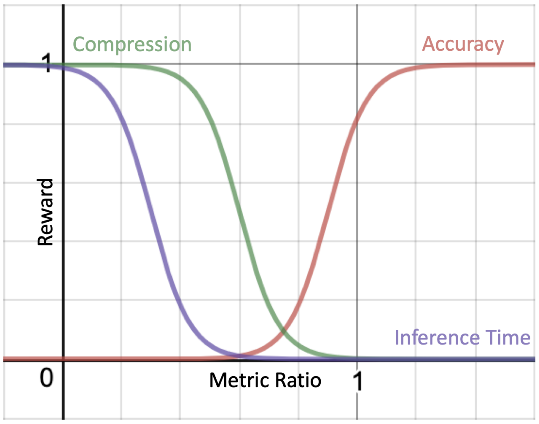

Fig. 1. Plot showing the reward structure of the 3 performance metrics

while still having realistic size and inference latency. The next section describes in detail the construction of the reward function R.

3.3 Reward Function

An intelligently designed reward function which can differentiate between good and bad architectures is necessary for effective exploration in the architectural space. We introduce threshold-ed reward function which incorporates user specific device restriction, this allows us to reinforce the policy network to learn architectures which fit the device restrictions. Our objective is obtaining architectures which have high accuracy, low memory footprint as well as faster and computationally inexpensive at test time. Models having lower inference time (test time) and reasonably high accuracy are preferred over models with very high accuracy (in particular very deep models) which have higher inference time. Unlike previous work which optimize for indirect metrics such as FLOPS [13], we consider the inference latency on a single GPU as the measure of inference latency. We formulate a reward which is a combination three metrics for a given model m: Accuracy $A(m)$, Compression percentage $C(m)$ and Inference time $T(m)$ (latency of model m) . We incorporate user specific device restrictions by introducing thresholds on Accuracy $A(m)$ and Latency $T(m)$. Incorporating real time latency in our reward allows us to effective search through the architectural space, resulting in model have high compression and reasonably high accuracy with the added benefit of lower inference time (which is essential element to be considered while deployment of the models to mobile devices) We perform intuitive transformation on accuracy Accuracy $A(m)$, Compression percentage $C(m)$ and latency $T(m)$ which are discussed in detail below.

Accuracy $A(m)$: Our primary objective is to obtain architectures which give high performance. One way to achieve this is by increasing the depth of the network [12, 25] but this comes at the expense of latency. We transform the reward obtained from accuracy such that we obtain a higher reward for model having high accuracy and is continuous function which provides flexibility to incorporate thresholds A_{th} in the reward. Formally, reward is obtained through the following transformation. $R_1 \colon A \longrightarrow [0, 1]$, which normalizes as well as enforces threshold to prevent exploration in undesired region of the search space.

$$R_1(A) = 1 - \left(1 + \exp\left(15 \cdot \left(\frac{A}{A_{teacher}} - A_{th}\right)\right)\right)^{-1} \tag{3}$$

where $A_{teacher}$ is accuracy of the teacher model.

Latency $T(m)$ **:** Inference Latency is essential component to considered which searching for a architecture fit for deployment. We incorporate the threshold by using a *shifted sigmoid* transformation which is defined as :

$R_2 \colon T \longrightarrow [0, 1]$:

$$R_2(T) = \left(1 + \exp\left(15 \cdot \left(\frac{T}{T_{teacher}} - T_{th}\right)\right)\right)^{-1} \tag{4}$$

Compression $C(m)$ **:** We use the ratio of the number of parameters of student and the teacher model and define the compression index as :

$C(m) = \#parameter(student)$, then similarly transform it to enforce our thresholds on size.

$$R_3(C) = \left(1 + \exp\left(15 \cdot \left(\frac{C}{C_{teacher}} - C_{th}\right)\right)\right)^{-1} \tag{5}$$

Thresholds are needed as we search for the right balance in AMS trade-off Space the thresholds are $A_{th} = 0.9$ $T_{th} = 0.3$ $C_{th} = 0.6$ are fixed for all of our experiments. Results pertaining to changing thresholds value are provided in Appendix-**??**. As *models with high degree of high compression does not guarantee lower inferences time.* The reward for the three performance metrics is given in 1 The final reward which need to be maximized for the RL process is defined as:

$$R(m) = R_1(A(m)) \cdot R_2(T(m)) \cdot R_3(C(m)) \tag{6}$$

3.4 Optimization

The parameters of the policy network characterized by θ are optimized to obtain a efficient policy to compress the teacher model. The optimization is formulated to maximize the *expected reward* obtained from the newly compressed architecture defined as :

$$\theta^* = \arg\max_{\theta} E_{(\boldsymbol{s},\boldsymbol{a}) \sim p_{\theta(\boldsymbol{s},\boldsymbol{a})}}(R(\boldsymbol{s})) \tag{7}$$

$$J(\theta) = E_{(\boldsymbol{s},\boldsymbol{a}) \sim p_{\theta(\boldsymbol{s},\boldsymbol{a})}}(R(\boldsymbol{s})) \tag{8}$$

where $R(\boldsymbol{s})$ is total reward obtained. The optimization process can be estimated using REINFORCE policy gradient used in [29], the continuous nature of our proposed reward transformation improves our search efficiency as it prevents exploding gradient. The gradient is estimated as:

$$\nabla_\theta J(\theta) = \nabla_\theta E_{(\boldsymbol{s},\boldsymbol{a})\sim p_{\theta(\boldsymbol{s},\boldsymbol{a})}}(R(\boldsymbol{s})) \tag{9}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log p_\theta\big(a_{i,t}, s_{i,t}\big)(R(\boldsymbol{s})) \tag{10}$$

here N represents number of produced architectures, T represents the length of trajectory. The above equation has high variance, to normalize that we utilize exponential moving average of the previous rewards as the baseline b subtract it from our total reward $R(\boldsymbol{s})$.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log p_\theta\big(a_{i,t}, s_{i,t}\big)(R(\boldsymbol{s}) - b) \tag{11}$$

This helps improve stability of the estimated gradients.

3.5 Knowledge Distillation

Student model architectures are trained utilizing both the outputs of the teacher models and the true label. Instead of just using the un-normalized log probabilities (logits) of the teacher model, which outperforms the training process used in [2]. Training incorporating dark knowledge [14] that helps student to mimic the relationships learned by the teacher model. The loss function is trained as combination of *hard* and *soft* targets, giving higher priority to transferring the dark knowledge. If y_i are output logits of the teacher model of the i^{th} training example, y_{true} is the true labels. Then the loss function is described below as:

$$\mathcal{L} = \lambda \cdot \mathcal{L}_{soft} + (1-\lambda)\cdot \mathcal{L}_{hard} \tag{12}$$

$$\begin{aligned}\mathcal{L}_{soft} &= D_{KL}(f(x;W) \,\|\, y_{true}) \\ &= \sum_i f(x^i;W)\cdot \log\frac{f(x^i;W)}{y_{true}(i)}\end{aligned} \tag{13}$$

$$\mathcal{L}_{hard} = H(f(x;W), y) = -\sum_i f(x^i;W)\cdot \log y_i \tag{14}$$

Through experimentation we have fixed value of $\lambda = 0.7$ thus making the student model to mimic the behaviour of the teacher model simultaneously fine-tuning the student architecture towards the true labels.

3.6 Transferring Learned Compression Policy

Unlike in the previous literature [31, 32] of transferring knowledge between different architecture families, we show transferability of the learned parameters of the policy across different subsets of data from a policy learned on the entire dataset . This not only provides a *warm start* to the policy network but also improves upon the time to converge to good model architectures for given dataset (up to 5x reduction in time). Hence allowing us to get high performing compressed model architectures which satisfy user specific device thresholds, owing to the efficacy of transferring learnt information across dataset. Furthermore, we can train a policy for a larger dataset and subsequently fine tune the policy in a small data environment (often the case with data on mobile devices) to produce good architectures for the user tailored to the data.

4 EXPERIMENTAL RESULTS

We explain the relevant details of the experimental setup below. In the spirit of reproducibility, we have made all datasets, implementations for experiments, and results available[1].

4.1 Datasets

We use the CIFAR-10 and CIFAR-100 [19] datasets for our experiments. Both these datasets have 50,000 training images and 10,000 test images of size 32x32. They diff in the number of labelled classes, 10 vs 100. We will also work with specific subsets of these datasets (consisting of smaller number of class labels) for some of the experiments. We consider the following subsets:

(1) `Animals`: Subset of CIFAR-10 formed by the class labels `bird`, `cat`, `deer`, `dog`, `frog` and `horse`.
(2) `Vehicles`$_{10}$: Subset of CIFAR-10 formed by the class labels `airplane`, `automobile`, `ship` and `truck`.
(3) For CIFAR-100, we use the superclasses as mentioned in the dataset. We use the superclasses `Insects`, `Fruits`, `Trees`, `Vehicles-1`, `Vehicles-2`, `People`, `Reptiles`.

4.2 Methods Under Study

(1) `Prun`: We compare our method to the popular ranking based pruning method [21]. We remove 512 filters on each iteration of pruning heuristically followed by 10 epochs of fine-tuning to recover the network.
(2) `KD`: Another popular approach for model compression is Knowledge Distillation. It requires a specific student architecture to train. We use a 7 layer deep CNN architecture inspired by the VGG architecture [25] for the student architecture.

[1] http://bit.ly/2BTduY5

Table 2. Compressed Model Search for VGG11 on Cifar10

Method	Accuracy	Compression	Inference Time (Sec)
Original Model	0.9	1x	3.43
Pruning	0.84	5x	1.96
Knowledge Distillation	0.79	2x	1.13
Auto-Keras	0.93	0.33x	7.9
N2N	**0.85**	9x	1.49
DDC$_{ht}$	0.81	**20.8x**	1.13
DDC	0.84	**20.8x**	**1.11**

(3) N2N: This method systematically finds a compressed optimal architecture by searching within the teacher's architecture [2]. The number of reinforcement learning iterations have been fixed at 100. In each iteration, 5 new student architectures are being trained using Knowledge distillation.

(4) AK: Auto-Keras [18] is a popular tool for architecture search which uses Bayesian optimization and searches a model from scratch given a dataset.

(5) DDC: This is our proposed method.

4.3 Implementation details

Our experiments provide evidence towards efficacy of imposing thresholds on reward to improve the architecture search. One important point to note here is that we do not use any proxies for inference time (like FLOPS) but instead we use actual inference time of the model on the system. For the memory footprint, we use number of parameters present in the model. Furthermore for consistency, we use the same teacher models across methods to provide evidence towards efficacy of our system. We use stochastic gradient descent with $momentum = 0.9$ and $learningrate = 0.001$ for all our experiments. Unless otherwise mentioned the number of reinforcement learning iterations is 100 and in each iteration we sample 5 student architecture according to the RL policy and train each of these student architectures for 20 epochs.

4.4 Compressed Model Search

In this set of experiments, we compress VGG11 and RESNET18 models for a good AMS trade-off on the entire dataset. We try all the methods detailed under Section 4.2. We consider 3 different combinations of architecture and dataset, *viz.* VGG11 with CIFAR10, RESNET18 with CIFAR10, and RESNET18 with CIFAR100. In order to study the effect of soft targets on the AMS trade-off, we also learned the compression policy (DDC_{ht}) by training the student architecture with hard targets only during knowledge distillation. The results are in Tables 2, 3 and 4.

Table 3. Compressed Model Search for ResNet18 on Cifar10

Method	Acc.	Comp.	Inf. Time (sec)
Original Model	0.83	1x	5.46
Knowledge Distillation	0.80	1.67x	3.12
Auto-Keras	0.93	0.40x	7.9
N2N	**0.81**	4.34x	1.60
DDC$_{ht}$	**0.81**	2.56x	1.69
DDC	**0.81**	**5.71**x	**1.48**

Table 4. Compressed Model Search for ResNet18 on Cifar100

Method	Acc.	Comp.	Inf. Time (sec)
Original Model	0.72	1x	2.71
Auto-Keras	0.68	0.40x	6.2
N2N	0.58	4.1x	1.56
DDC$_{ht}$	**0.63**	2.17x	1.68
DDC	0.56	**4.4x**	**1.40**

The results indicate that our method is able to find highly compressed models that have low inference time without compromising much on the accuracy. Introduction of inference time as a metric in reward function not only helps the compression policy to find faster models but also helps the policy to achieve a much better accuracy-compression trade-off. In case of VGG11 with CIFAR10, our method is able to find a compressed model that is 20.8x times smaller than the teacher model with a drop of 6% accuracy. Also the produced compressed model is 3 times faster that the original model. We see that the accuracy tradeoff of compressed models in case of CIFAR100 is more when compared to models compressed on CIFAR10 dataset. This may be because CIFAR100 is a much harder dataset than CIFAR10. It has 100 classes with fewer samples per class than the CIFAR10 dataset. we also observe that the models produced by auto-keras though have good accuracy, are much larger that the original model thereby making them unusable for the task of model compression.

4.5 Subset based Compression

If high accuracy is needed only on a subset of the class labels, it is possible to imagine that much better inference speed and memory footprints may be possible post-compression. In subset based compression we learn the RL policy by training the student architectures sampled in each RL iteration, only on the desired subsets rather than on the entire dataset. This will enable our

Table 5. ResNet18 AMS Trade-off of DDC on full vs partial CIFAR100 dataset

Subset	Policy	Acc.	Comp.	Inf. Time (sec)
fruits	Full	0.64	4.34x	0.12
	Sub	**0.73**	**6.3x**	**0.096**
insects	Full	0.61	**4.3x**	0.12
	Sub	**0.67**	3.3x	**0.087**
people	Full	0.36	4.3x	0.11
	Sub	**0.388**	**6.3x**	**0.064**
reptiles	Full	0.32	4.3x	0.11
	Sub	**0.556**	**4.7x**	**0.069**
trees	Full	0.57	**4.3x**	0.11
	Sub	**0.606**	3.4x	**0.099**
vehicles-1	Full	**0.67**	**4.3x**	0.13
	Sub	0.664	2.3x	**0.079**
vehicles-2	Full	0.57	4.3x	0.13
	Sub	**0.765**	**6.3x**	**0.064**

Table 6. ResNet18 AMS Trade-off of DDC on full vs partial CIFAR10 dataset

Subset	Policy	Acc.	Comp.	Inf. Time (sec)
animals	Full	0.79	**5.7x**	0.99
	Sub	**0.815**	4.14x	**0.913**
vehicles$_{10}$	Full	0.85	5.71x	0.703
	Sub	**0.89**	**5.89x**	**0.549**

Table 7. VGG11 AMS Trade-off of DDC on full vs partial CIFAR10 dataset

Subset	Policy	Acc.	Comp.	Inf. Time (sec)
animals	Full	**0.79**	**20x**	0.862
	Sub	**0.79**	14.3x	**0.75**
vehicles$_{10}$	Full	**0.9**	20x	0.623
	Sub	0.87	**50x**	**0.415**

compression method to learn subset specific policy that has a better AMS tradeoff when compared to the policy learnt on the entire dataset.

Tables 5, 6 and 7 compares the performance of the compressed models produced by policies learnt on subsets with the compressed model produced by the policy learnt on the entire dataset. The results indicate that the models produced by the polices learnt on the subsets outperforms the model produced by the policy learned on the entire dataset on all 3 performance metrics for most of the subsets. The enhancement in performance is clearly evident in the case of CIFAR-100 subsets. The policy learns to remove layers of the teacher model who's learned features doesn't help in differentiating samples from the subsets thus, giving a better AMS tradeoff.

4.6 Transfer of Compression Policy

The previous experiment demonstrates that subset based model compression helps in achieving a better AMS tradeoff. But datasets for practical applications are often huge with large number of classes. Finding compressed policy for all the subsets of classes will be prohibitive. In terms of concrete numbers, learning a policy for the animals subset of CIFAR-10 using RESNET18 as teacher model takes around 12 hours for DDC. We propose compression policy transfer as a way to scale our algorithm for large scale deployments. In policy transfer, we use the compression policy learnt on the entire dataset to bootstrap the learning of the policy for the subsets. We observed that the compression policy learnt by policy transfer is able to produce compressed models with a good AMS tradeoff in around 20 RL iterations.

Tables 8, 9 and 10 tabulates the performance of the compressed models produced by policies learnt from scratch and policies learnt from policy transfer after 20 RL iterations. The performance of the policy transfered models after 20 epoch are comparable to policy learnt from scratch models after 100 epoch.

5 CONCLUSION

This paper presents an automated systematic approach to compress a convolutional neural networks using Reinforcement learning. The key idea behind this method is to incorporate real-world latency information into reward function to find faster performing compressed models without much tradeoff in accuracy. We demonstrated that subset based model compression can be helpful in scenarios where the deployed model sees only a subset of data classes. We proposed policy transfer as way to solve large scale deployment of our compression algorithm.

REFERENCES

[1] 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), 32.

[2] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. 2017. N2N Learning: Network to Network Compression via Policy Gradient Reinforcement Learning. *arXiv preprint arXiv:1709.06030* (2017).

Table 8. ResNet18 AMS Trade-off of DDC on policy transfer vs learning from scratch on subsets of CIFAR100 dataset after 20 epochs

Subset	Policy	Acc.	Comp.	Inf. Time (sec)
fruits	Transfer	**0.748**	**6.3x**	**0.062**
	Scratch	0.716	4.3x	0.109
insects	Transfer	0.646	**6.3x**	**0.063**
	Scratch	**0.66**	2x	0.079
people	Transfer	**0.393**	**6.3x**	**0.066**
	Scratch	0.354	4.7x	0.073
reptiles	Transfer	**0.566**	**6.3x**	**0.065**
	Scratch	0.534	2x	0.081
trees	Transfer	**0.592**	**4.7x**	**0.067**
	Scratch	0.568	3.4x	0.085
vehicles-1	Transfer	**0.678**	**4.7x**	**0.069**
	Scratch	0.62	1.7x	0.089
vehicles-2	Transfer	**0.79**	**6.3x**	**0.063**
	Scratch	0.714	4.3x	0.083

Table 9. ResNet18 AMS Trade-off of DDC on policy transfer vs learning from scratch on subsets of CIFAR10 dataset after 20 epochs

Subset	Policy	Acc.	Comp.	Inf. Time (sec)
animals	Transfer	0.783	**6.45x**	**0.801**
	Scratch	**0.829**	3.5x	0.968
vehicles_{10}	Transfer	0.888	**6.45x**	**0.496**
	Scratch	**0.896**	5.8x	0.580

[3] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep?. In *Advances in neural information processing systems*. 2654–2662.

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

[5] Shengcao Cao, Xiaofang Wang, and Kris M Kitani. 2019. Learnable embedding space for efficient neural architecture compression. *arXiv preprint arXiv:1902.00383* (2019).

[6] An-Chieh Cheng, Jin-Dong Dong, Chi-Hung Hsu, Shu-Huan Chang, Min Sun, Shih-Chieh Chang, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Da-Cheng Juan. 2018. Searching Toward Pareto-Optimal Device-Aware Neural Architectures. *arXiv preprint arXiv:1808.09830* (2018).

Table 10. VGG11 AMS Trade-off of DDC on policy transfer vs learning from scratch on subsets of CIFAR10 dataset after 20 epochs

Subset	Policy	Acc.	Comp.	Inf. Time (sec)
animals	Transfer	**0.81**	**20.83x**	**0.69**
	Scratch	0.81	7.57x	0.732
vehicles_{10}	Transfer	0.880	**43.4x**	**0.423**
	Scratch	**0.894**	7.81x	0.425

[7] 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). arXiv:1710.09282 http://arxiv.org/abs/1710.09282

[8] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Multi-objective architecture search for cnns. *arXiv preprint arXiv:1804.09081* (2018).

[9] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*. 1379–1387.

[10] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 243–254.

[11] 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).

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[13] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 784–800.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[16] Jie Hu, Li Shen, and Gang Sun. 2017. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507* 7 (2017).

[17] 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< 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).

[18] Haifeng Jin, Qingquan Song, and Xia Hu. 2018. Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282* (2018).

[19] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images.* Technical Report. Citeseer.

[20] Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*. 598–605.

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

[22] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2018. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548* (2018).

[23] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. 2017. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041* (2017).

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

[25] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[26] Suraj Srinivas and R Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149* (2015).

[27] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning.. In *AAAI*, Vol. 4. 12.

[28] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. 2018. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *arXiv preprint arXiv:1807.11626* (2018).

[29] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[30] Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arık, Haonan Yu, Hairong Liu, and Greg Diamos. 2018. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912* (2018).

[31] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[32] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2017. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012* 2, 6 (2017).