Supplementary material for: Low-rank compression of neural nets: learning the rank of each layer

Yerlan Idelbayev   Miguel Á. Carreira-Perpiñán
Dept. of Computer Science and Engineering, University of California, Merced
http://eecs.ucmerced.edu

March 29, 2020

Abstract

This supplemental materials contain the following additional information: details of metrics computation (section 1); theoretical results on choosing μ-schedule (section 2); full details of all experiments with additional plots and extended tables: LeNets (section 4.1), VGGs (section 4.2), ResNets (section 4.3), NIN (section 4.4) and AlexNet (section 4.5); extended comparison to the networks trained on the CIFAR10 dataset (section 4.6).

1 Metrics computation

1.1 Ratios

The compression ratio ρ_{storage} is defined wrt the storage bits, in the following way:

$$\rho_{\text{storage}} = \frac{\texttt{bits}\,(\text{reference})}{\texttt{bits}\,(\text{compressed})} = \frac{\texttt{params}\,(\text{reference})}{\texttt{params}\,(\text{compressed})}, \tag{1}$$

here the ratio of bits is equivalent to the ratio of params since all low-rank structures can be directly expressed as a sequence of layers. Unless specifically mentioned, we apply simple parametrization involving $\mathbf{U}, \mathbf{V}$ matrices, and report corresponding compression ratios.

The ratio of FLOPs is defined in the similar way:

$$\rho_{\text{FLOPs}} = \frac{\text{FLOPs}\,(\text{reference})}{\text{FLOPs}\,(\text{compressed})}, \tag{2}$$

where the exact definition of FLOPs is given in the next subsection.

1.2 Number of floating point operations, FLOPs

There is no clear consensus in the literature on how to compute the total number of floating point operations, FLOPs[1], in the forward pass of a neural network. While some authors define this number as a total number of multiplications and additions (e.g., [22]), others count one multiplication and addition as a single operation (e.g., [4]), assuming multiplications and additions can be fused. In this work, we adopt the latter definition of FLOPs — the total number of *fused* multiplications and additions incurred by all layers in the network.

FLOPs in fully-connected layer For a fully-connected layer with weights $\mathbf{W} \in \mathbb{R}^{m\times n}$ and biases $\mathbf{b} \in \mathbb{R}^m$ the total number of FLOPs is

$$\text{fc-FLOPs}\,(\mathbf{W}, \mathbf{b}) = \underbrace{m \times (n-1)}_{\text{mult/ads in } \mathbf{Wx}} + \underbrace{m}_{\text{adds in } \mathbf{b}} = mn.$$

[1] The term MAC — multiply/accumulate counts is widely used too.

FLOPs in convolutional layer Each convolutional layer with parameters $\mathbf{W}, \mathbf{b}$ is a linear mapping applied multiple (M) times. Thus, FLOPs defined as:

$$\text{conv-FLOPs} = \text{fc-FLOPs}\,(\mathbf{W}, \mathbf{0}) \times M + \text{fc-FLOPs}\,(\mathbf{0}, \mathbf{b})$$

Our definition of FLOPs omits the batch-normalization (BN) and concatenation/copy operations. Here is how we treat them. For BN layers, the BN parameters can be fused into the weights and biases of the preceding layer, therefore no special treatment is required. For concatenations, copy operations, and nonlinearities we assume zero FLOPs due to its negligible cost.

2 On the μ-schedule and beginning of the path

Our algorithm belongs to the class of homotopy methods, and follows a path indexed by a user defined schedule over values of $\mu = \mu_0, \mu_1, \ldots,$ etc. (see pseudo code in the main paper). Particularly, the solution of the C-step for the layer k is given by:

$$\min_r \lambda\, C_k(r) + \frac{\mu}{2} \sum_{i=r+1}^{R_k} s_{ki}^2 \quad \text{s.t.} \quad r_k \in \{0, 1, \ldots, R_k\} \tag{3}$$

and traces a path $r(\mu)$ for every value of μ, and implicitly a path over $\mathbf{U}(\mu), \mathbf{V}(\mu)$. The beginning of the path is when $\mu = 0$, for which the solution is $r = 0$. Since the ranks are discrete valued, the selected rank $r = 0$ will be unchanged until some threshold of μ_k^+ is reached. The knowledge of this threshold helps us to come up with a better μ schedule, and especially with a choice of the initial value for μ_0.

The solution of (3) will be $r = 0$ as long as the rank-0 approximation is yielding a better loss than the rank-1 approximation, i.e., for all μ satisfying:

$$\lambda\, C_k(0) + \frac{\mu}{2} \sum_{i=1}^{R_k} s_{ki}^2 < \lambda\, C_k(1) + \frac{\mu}{2} \sum_{i=2}^{R_k} s_{ki}^2 \quad \Longleftrightarrow \quad \mu \le \mu_k^+ := \frac{2\lambda C_k(0)}{s_{k1}^2}. \tag{4}$$

Here we defined the threshold μ_k^+ by solving LHS. This result implicitly depends on μ-value itself, as s_{k1} is the largest singular value of the matrix $\mathbf{W}_k(\mu^+)$. However, if we assume $\mathbf{W}_k(\mu^+) \approx \mathbf{W}_k^{\text{init}}$, we can take $s_{k1} = \sigma_1(\mathbf{W}_k^{\text{init}})$, and compute the threshold μ_k^+ for layer k. Then we set as initial value μ_0 the smallest μ_k^+ across all layers:

$$\mu_0 = \min_k \mu_k^+ \tag{5}$$

In our experiments, we choose the value for μ_0 to be greater or equal to the result suggested by (5).

3 Baselines for rank selection

We adopt the following baselines proposed in the literature to compare to our method.

Baseline 1 [24] proposed to estimate the reduced ranks of the weight matrices by solving the following minimization problem over the ranks:

$$\max_{r_1, \ldots, r_l} \prod_l \sum_{i=1}^{r_l} \sigma_{l,i} \quad \text{s.t.} \quad \sum_l \frac{r_l}{\text{rank}\,(\mathbf{W}_l)} C_l \le p \cdot C, \tag{6}$$

where $\sigma_{l,i}$ is is i-th singular value of the layer l, C_l is the number of FLOPs in the forward pass through the layer l, C is the total number of FLOPs required for the forward pass, and $p \in (0, 1)$ is a user provided parameter that allows to obtain various sized models. This optimization problem maximizes the accumulated energy subject to a complexity constraint, and is solved greedily by removing one rank at a time from a layer which reduces the objective function (6) the least, starting from full ranks $r_i = \text{rank}\,(\mathbf{W}_i)$.

Baseline 2 Even simpler heuristic is to choose the rank r_i such that r_i-rank approximation is within p-ratio of the original matrix, i.e., the highest rank satisfying $\|\mathbf{UV}\|_F \leq p\|\mathbf{W}\|_F$. For example, [20] use $p = 0.95$ and [21] use both $p = 0.95$ and $p = 0.99$. By varying this ratio p we can obtain different sized models, which we fine-tune and report results.

4 Experiments

This section contains full details of all the experiments on our rank selection method and baselines, and supplements it with additional plots and extended tables. Our experiments and the baselines are run in the following way:

$$\text{rank selection} \longrightarrow \text{re-parametrization} \longrightarrow \text{fine-tuning.}$$

Here, the re-parametrization is a process of converting low-rank matrices to a sequence of fully-connected or convolutional layers. If during the re-parametrization step some of the ranks are not efficient, i.e., leading to more storage or computation depending on optimization criterion $C(r)$, we leave such layers as a full rank without changing its parametrization. We report the compression ratios of memory and FLOPs with respect to the final re-parametrized version of a neural network. For experiments reported in this suppl. mat. we use the computational cost criterion $C(r) = \text{FLOPs}\,(r) \times 10^{-6}$, i.e., defined wrt MFLOPS.

4.1 LeNet-s on MNIST

We train LeNet300, LeNet5 [10] on MNIST dataset (10 classes, 60k grayscale images of 28×28). The full details of these architectures can be found in Table 1. Images are first normalized to have grayscales between 0 and 1 and then the mean image was subtracted. We report results obtained at the end of the training.

Reference nets are trained with Nesterov's accelerated SGD [16] with momentum of 0.9 on minibatches of size 256. The loss is average cross entropy. Networks are trained for 300 epochs with an initial learning rate of 0.1 decayed by 0.99 after every epoch. The resulting test errors are 1.98% for LeNet300 and 0.55% for LeNet5.

Rank learning via *our algorithm* is run for 30 LC iterations, with $\mu = 0.001 \times 1.1^k$ at k-th iteration for both LeNet300 and LeNet5. Each L-step is performed by Nesterov's SGD with momentum 0.9 and run for 30 epochs with an initial learning rate of 0.1 decayed by 0.98 after each L-step. Fine-tuning is performed on the network which directly parametrize low rank as a sequence of fully-connected or convolutional layers, with Nesterov's SGD with momentum 0.9 for 100 epochs and an initial learning rate of 0.02 decayed by 0.99 after each epoch. *Runtime.* Training time of our networks is 3× of the training time of the reference net.

LeNet300

Layer	Connectivity
Input	28×28 image
1	fully connected, 300 neurons, followed by tanh
2	fully connected, 100 neurons, followed by tanh
3 (output)	fully connected, 10 neurons, followed by softmax
$P_1 = 266\,200$ weights, $P_0 = 410$ biases	

LeNet5

Layer	Connectivity
Input	28×28 image
1	convolutional, 20 5×5 filters (stride=1), total 11 520 neurons, followed by ReLU
2	max pool, 2×2 window (stride=2), total 2 280 neurons
3	convolutional, 50 5×5 filters (stride=1), total 3 200 neurons, followed by ReLU
4	max pool, 2×2 window (stride=2), total 800 neurons
5	fully connected, 500 neurons and dropout with $p = 0.5$, followed by ReLU
6 (output)	fully connected, 10 neurons and dropout with $p = 0.5$, followed by softmax
$P_1 = 430500$ weights, $P_0 = 580$ biases	

Table 1: Structure of the LeNet300 and LeNet5 neural nets trained on the MNIST dataset.

	$\lambda \times 10^{-3}$	ranks	params	FLOPs	Before fine-tuning			After fine-tuning			ρ_{FLOPs}	ρ_{storage}
					$\log L$	E_{train}	E_{test}	$\log L$	E_{train}	E_{test}		
LeNet300	**R**	[300 100 10]	0.26M	0.26M	-3.68	0.00	1.98				1.00	1.00
	.25	[35 16 9]	46 150	45 330	-3.45	0.17	1.88	-4.11	0.00	1.87	5.87	5.77
	.50	[28 13 9]	37 362	36 542	-3.39	0.47	1.99	-4.06	0.00	2.00	7.28	7.12
	.75	[25 12 9]	33 710	32 890	-3.35	1.87	2.05	-4.03	0.00	2.05	8.09	7.90
	1	[24 10 9]	31 826	31 006	-3.32	4.39	2.03	-4.02	0.00	2.06	8.59	8.36
	2	[18 9 9]	24 922	24 102	-2.98	8.47	2.38	-3.88	0.00	2.39	11.04	10.68
	3	[16 8 9]	22 354	21 534	-2.37	13.89	2.77	-3.78	0.00	2.72	12.36	11.91
	4	[10 7 9]	15 450	14 630	-0.50	14.89	10.34	-2.04	0.13	4.58	18.20	17.23

Figure 1: Detailed table of our experiments on the LeNet300. We report: training loss $\log L$ and training and test classification error E_{train} and E_{test} (%); reduction of FLOPs (ρ_{FLOPs}) and parameters (ρ_{storage}).

	$\lambda \times 10^{-3}$	ranks	params	FLOPs	Before fine-tuning			After fine-tuning			ρ_{FLOPs}	ρ_{storage}
					$\log L$	E_{train}	E_{test}	$\log L$	E_{train}	E_{test}		
LeNet5	**R**	[20 50 500 10]	0.43M	2.29M	-6.27	0.00	0.55				1.00	1.00
	0.4	[5 5 14 9]	26 855	328 390	-2.55	0.17	1.00	-3.73	0.01	0.75	6.98	16.04
	1.0	[4 5 9 9]	20 310	295 970	-0.42	0.47	11.18	-3.27	0.01	0.82	7.75	21.20
	1.1	[3 3 9 9]	19 165	199 650	0.25	1.87	43.79	-2.28	0.16	1.33	11.49	22.47
	1.4	[2 3 9 9]	19 120	173 730	0.27	4.39	57.79	-2.14	0.21	1.44	13.20	22.52
	2.0	[2 2 9 9]	18 570	138 530	0.33	8.47	67.57	-1.75	0.57	1.66	16.55	23.19
	2.4	[2 1 8 9]	16 720	102 030	0.37	13.89	77.52	-1.11	2.48	2.90	22.47	25.76
	3.0	[1 1 7 9]	15 375	74 810	0.59	14.89	88.65	-0.98	3.39	3.80	30.65	28.01

Figure 2: Detailed table of our experiments on the LeNet5. We report: training loss $\log L$ and training and test classification error E_{train} and E_{test} (%); reduction of FLOPs (ρ_{FLOPs}) and parameters (ρ_{storage}).

4.2 VGGs on CIFAR10

We train the VGG16 and VGG19 [18] adapted for the CIFAR10 dataset (10 classes, 60k RGB images of 32×32). We employ batch normalization after every layer except the last, and dropouts after fully connected layers (see Table 2 for the full details). Images in the dataset are normalized channel wise to have zero mean and variance one. For training, we use simple augmentation (random horizontal flip, zero pad with 4 pixels on each side and randomly crop 32×32 image). For test we use normalized images without augmentation. We report results obtained at the end of the training. The loss is average cross entropy with ℓ_2 weight decay. The resulting nets have 15M (VGG16) and 20M (VGG19) parameters, and require 313 MFLOPs and 399 MFLOPs respectively.

Reference nets are trained with Nesterov's accelerated SGD [16] with momentum of 0.9 on minibatches of size 128. The loss is average cross entropy with a weigth decay of 5×10^{-4}. Networks are trained for 300 epochs with an initial learning rate of 0.05 decayed by 0.97716 after every epoch. The resulting test errors are 6.57% (VGG16) and 6.47% (VGG19).

Rank learning via *LC algorithm* is run for 60 LC iterations, with $\mu = 2 \times 10^{-5} \times 1.2^k$ at k-th iteration. Each L-step is performed by Nesterov's SGD with momentum of 0.9 and run for 15 epochs with learning rate of 0.07 at the beginning of the step and decayed by 0.99 after each epoch. Fine-tuning is performed on the network which directly parametrize low rank as a sequence of fully-connected or convolutional layers with Nesterov's SGD with momentum 0.9 for 300 epochs, with the initial learning rate 7×10^{-4} and decayed by 0.99 after each epoch. *Training time.* Running the LC algorithm with fine-tuning is 3 times longer comparing to the training of the reference network. The results are presented in Table 3.

Baselines. As discussed in Section 3, we train models with two different baselines. We remind in passing that *baseline 1* is due to Zhang et al. [24], and *baseline 2* is a simple singular value thresholding scheme used across the literature. Every baseline is controlled by a single hyperparameter, by varying which we

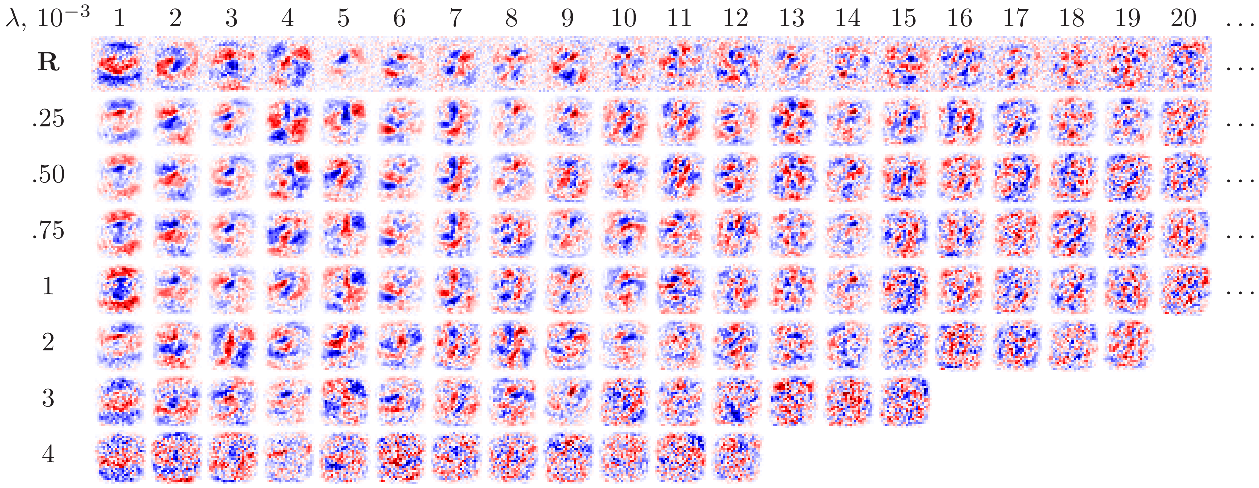

Figure 3: Neurons of the first layer of the compressed LeNet300 looking at an input image for different λ values, thus different compression levels; Each column i in the table corresponds to a reshaped i-th row of $\mathbf{V}$ in the order of decreasing singular values (from left to right). As you can see, neurons corresponding to the larger singular values (left) behave as smooth, low-pass filters and neurons corresponding to the smaller singular values (right) shows more oscillatory, high-pass filter behavior. As compression level increases (to bottom), neurons become noisier. Every image has been normalized to have maximum intensity of 1. Red: positive weights, blue: negative, white: zero, up to first 20 neurons are shown.

obtain different low-rank neural networks. Each obtained network is fine-tuned using Nesterov's SGD for 600 epochs with the initial learning rate of 0.001, which is decayed by 0.99 after every epoch. The comparison between our algorithm and the baselines is given in Figure 4.

Selected ranks We plot selected ranks for each layer of the VGG16 and VGG19, and corresponding FLOPs of these decompositions obtained by running our algorithm, see Figure 5. We see that selected ranks are not uniform at all, and some layers, e.g., the layers 5 and 9 of VGG16, have much higher ranks comparing to others. Most importantly, their relative proportion does not stay the same for different λ values. Take a look at the layers 10 and 11 of the VGG16 in Figure 5, for the value of $\lambda = 0.5$ the selected rank of layer 10 is greater than of layer 11, but for a higher value of $\lambda = 0.8$, the relation is reversed. These relations can not be captured by simple heuristics, and need to be inferred in an optimal way.

The ranks of convolutional layers do not correspond exactly to its contributed FLOPs, as each filter gets applied multiple times to the same input. Therefore, the layers with the higher ranks are not the ones with

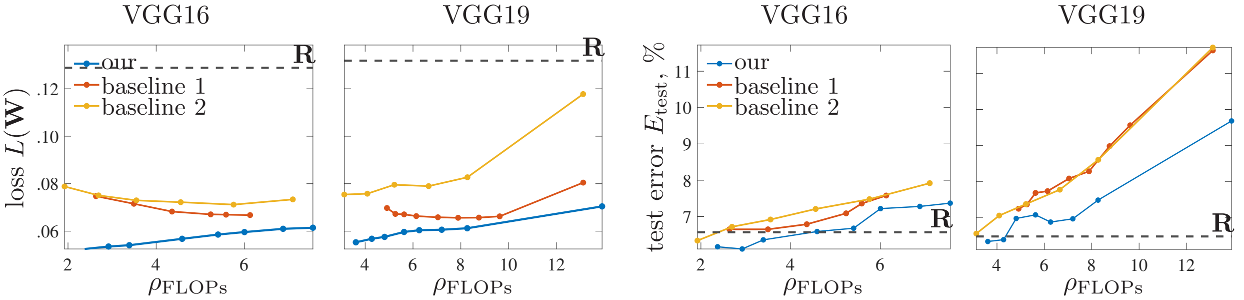

Figure 4: Comparison of our rank selection algorithm to the baselines on VGG16 and VGG19. *Baseline 1* is due to Zhang et al. [24], and *baseline 2* is a simple singular value thresholding scheme used across the literature. Left figures: loss vs FLOPs reduction ratio (ρ_{FLOPs}), right figures: test error vs ρ_{FLOPs}. As you can see our algorithm achieves both lower loss and better accuracy throughout. For details of training refer to text.

higher computational cost, e.g., see the layers 4 and 5 of VGG16. In general, VGG architecture is considered to be highly over-parametrized, which we empirically confirm by reducing its computational workload by factors of 4–6× and number of parameters by 5–9× without any degradation in test accuracy.

Layer	Connectivity
Input	32×32 image
1	convolutional, 64 3×3 filters (stride=1), followed by BN and ReLU
2	convolutional, 64 3×3 filters (stride=1), followed by BN and ReLU
	max pool, 2×2 window (stride=2)
3	convolutional, 128 3×3 filters (stride=1), followed by BN and ReLU
4	convolutional, 128 3×3 filters (stride=1), followed by BN and ReLU
	max pool, 2×2 window (stride=2)
5	convolutional, 256 3×3 filters (stride=1), followed by BN and ReLU
6	convolutional, 256 3×3 filters (stride=1), followed by BN and ReLU
7	convolutional, 256 3×3 filters (stride=1), followed by BN and ReLU
	max pool, 2×2 window (stride=2)
8	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
9	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
10	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
	max pool, 2×2 window (stride=2)
11	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
12	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
13	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
	max pool, 2×2 window (stride=2)
14	fully connected, 512 neurons and dropout with $p = 0.5$, followed by ReLU
15	fully connected, 512 neurons and dropout with $p = 0.5$, followed by ReLU
16 (output)	fully connected, 10 neurons, followed by softmax
14981952 weights, 8970 biases, 8448 running means/variances for BN	

Table 2: Structure of the adapted VGG16 network for the CIFAR10 dataset. BN–Batch Normalization, ReLU – rectified linear units. The VGG19 is adapted in a similar way.

	$\lambda \times 10^{-4}$	# params	MFLOPs	Before fine-tuning			After fine-tuning			ρ_{FLOPs}	ρ_{storage}
		$\times 10^6$		$\log L$	E_{train}	E_{test}	$\log L$	E_{train}	E_{test}		
VGG16	**R**	14.9	313.46	-0.89	0.00	6.57				1.00	1.00
	0.50	3.81	132.19	-1.19	0.03	6.44	-1.28	0.00	6.17	2.37	3.93
	0.75	3.16	107.34	-1.18	0.02	6.70	-1.27	0.00	6.11	2.92	4.75
	1.00	2.75	92.44	-1.17	0.02	6.78	-1.27	0.00	6.36	3.39	5.44
	2.00	2.00	68.29	-1.11	0.10	7.24	-1.25	0.00	6.59	4.59	7.48
	3.00	1.70	58.00	-1.09	0.13	7.37	-1.23	0.00	6.68	5.40	8.84
	4.00	1.53	52.21	-1.07	0.17	7.92	-1.22	0.00	7.22	6.00	9.79
	6.00	1.32	45.56	-1.03	0.39	8.40	-1.21	0.00	7.28	6.88	11.31
	8.00	1.21	41.48	-0.99	0.69	8.26	-1.21	0.00	7.37	7.56	12.37
VGG19	**R**	20	398.39	-0.88	0.00	6.47				1.00	1.00
	1.00	3.92	104.63	-1.15	0.02	6.72	-1.26	0.00	6.33	3.81	6.95
	1.50	2.47	88.36	-1.12	0.08	7.30	-1.25	0.00	6.38	4.51	8.21
	2.00	2.21	78.55	-1.11	0.06	7.22	-1.18	0.02	6.97	5.07	9.19
	3.00	1.88	67.19	-1.07	0.18	7.94	-1.22	0.00	7.07	5.93	10.80
	4.00	1.69	60.36	-1.04	0.34	8.01	-1.22	0.00	6.87	6.60	12.01
	6.00	1.44	52.49	-1.00	0.61	8.28	-1.21	0.00	6.96	7.59	14.09
	8.00	1.28	45.72	-0.99	0.71	8.20	-1.26	0.01	7.48	8.71	15.86
	30.0	0.76	27.19	-0.24	17.07	19.10	-1.15	0.46	9.68	14.65	26.79

Table 3: Detailed table of our experiments on VGG16 and VGG19. We report: loss L, training and test classification error E_{train} and E_{test} (%); reduction in FLOPs wrt reference, ρ_{FLOPs}; and reduction in the number of parameters wrt reference, ρ_{storage}. Logarithms are base 10.

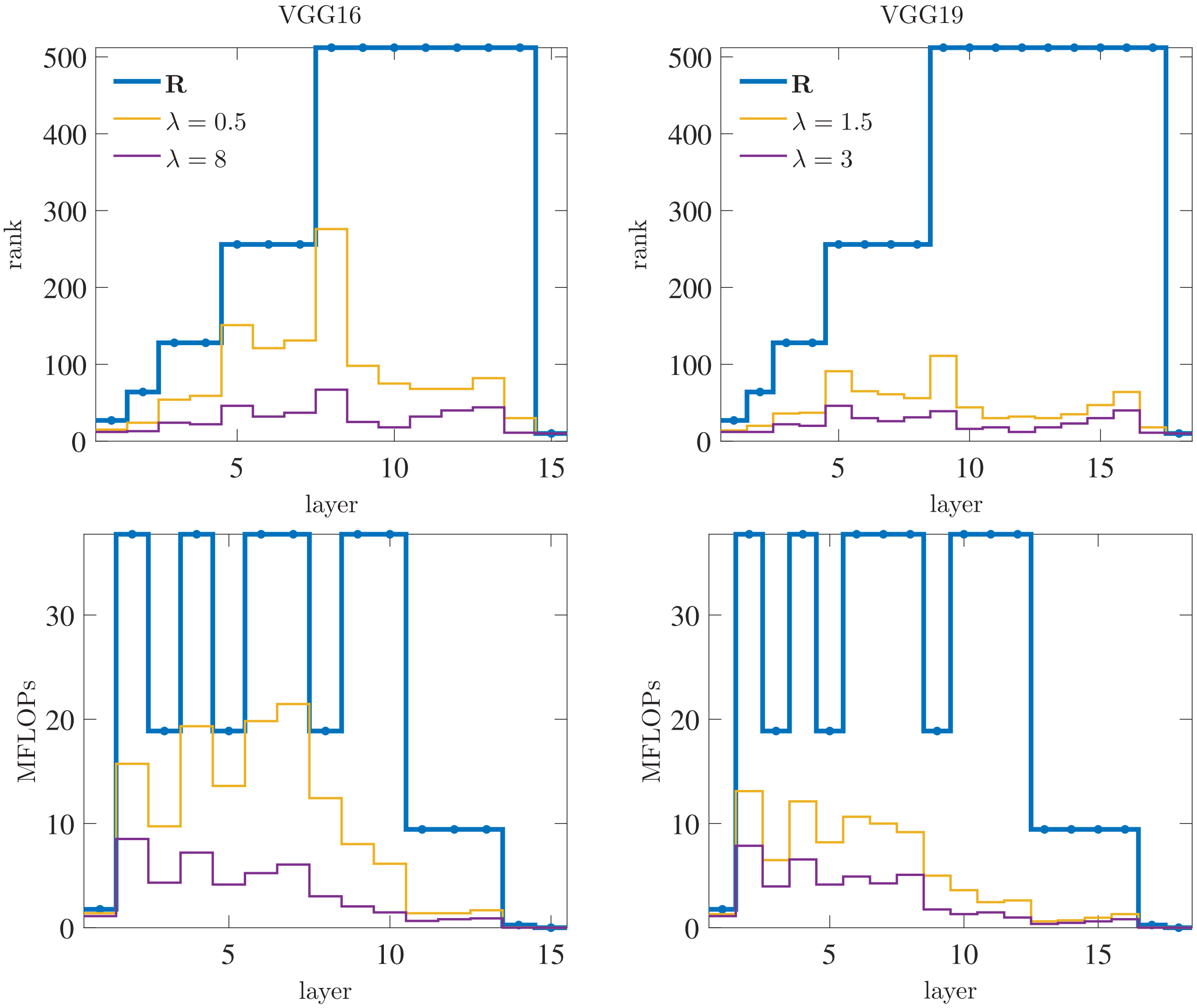

Figure 5: Some final selected architectures in terms of the rank of a layer and FLOPs of a layer for VGG16 and VGG19 using our method. For λ values, the multiplicative factor of $\times 10^{-4}$ is omitted.

4.3 ResNets on CIFAR10

We train ResNets [4] of depth 20, 32, 56 and 110 layers (0.27M, 0.46M, 0.85M, and 1.7M parameters, respectively) on the CIFAR10 dataset using the same augmentation setup as in [4]. Images in the dataset are normalized to have channel-wise zero mean, variance 1. For training, we use random horizontal flip, zero pad with 4 pixels on each side and randomly crop 32×32 image. For test we use normalized images without augmentation. We report results obtained at the end of the training. The loss is average cross entropy with a weight decay (as in the original paper). We train reference nets, nets compressed with our LC algorithm and with baselines, followed by fine-tuning of the resulting weights.

Reference nets are trained with Nesterov's SGD [16] with the momentum of 0.9 on minibatches of size 128. The loss is average cross entropy with a weight decay of 10^{-4}, weights are initialized following [3]. The network is trained for 200 epochs with learning rate of 0.1 which is divided by 10 after 100 and 150 epochs.

Our rank learning algorithm is run for 50 LC iterations, with $\mu = \times 10^{-3} \times 1.25^k$ at k-th iteration. Each L-step is performed by Nesterov's SGD with momentum of 0.9 and run for 15 epochs with learning rate of 0.001 at the beginning of the step and decayed by 0.99 after each epoch. Fine-tuning is performed on the network which directly parametrize low rank as a sequence of fully-connected or convolutional layers with Nesterov's SGD with momentum of 0.9 for 200 epochs, the initial learning rate of 7×10^{-4} which is decayed by 0.99 after each epoch. *Runtime.* Running the LC with fine-tuning is 3.75 times longer comparing to the training of the reference network.

Baselines. As discussed in Section 3, we train models with two different baselines. We remind in passing that *baseline 1* is due to Zhang et al. [24], and *baseline 2* is a simple singular value thresholding scheme used across the literature. Every baseline is controlled by a single hyperparameter, by varying which we obtain different low-rank neural networks. Each obtained network is fine-tuned using Nesterov's SGD with learning rate of 0.002 (with learning rate decay of 0.99 after every epoch) and momentum of 0.9 for 600 epochs. The comparison between our algorithm and baselines is given in Figure 6.

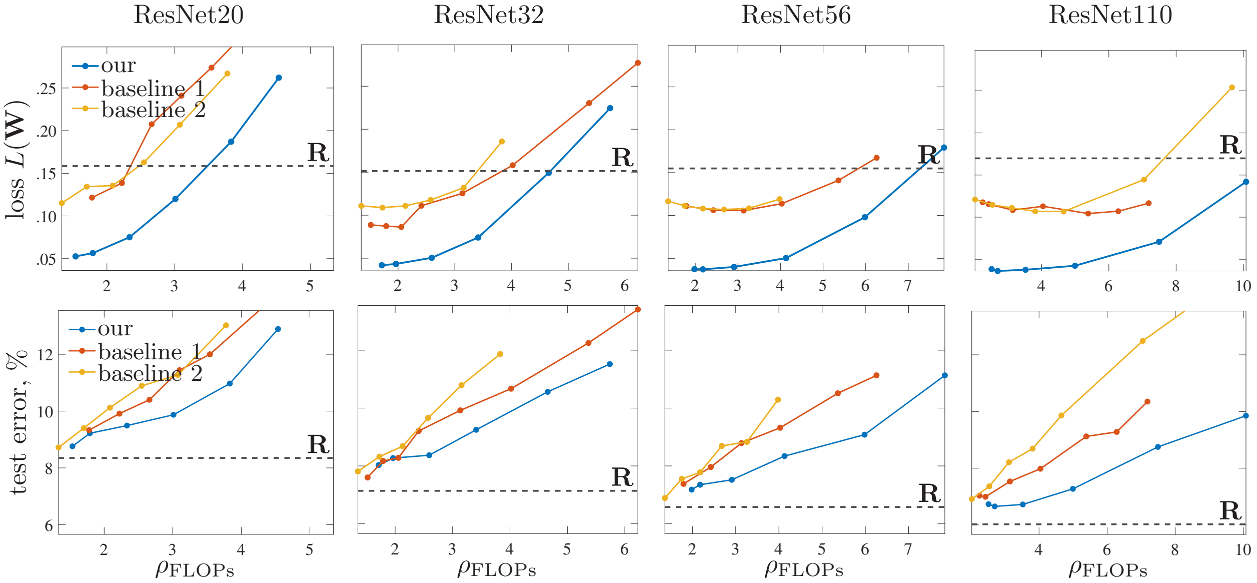

Figure 6: Comparison of our rank selection algorithm to the baselines on ResNet-s. *Baseline 1* is due to Zhang et al. [24], and *baseline 2* is a simple singular value thresholding scheme used across the literature. Top figures: loss vs FLOPs reduction ratio (ρ_{FLOPs}), right figures: test error vs ρ_{FLOPs}. As you can see our algorithm achieves both lower loss and better accuracy and fewer FLOPs throughout. For details of training refer to text.

	$\lambda \times 10^{-3}$	# params $\times 10^6$	MFLOPs	Before fine-tuning			After fine-tuning			ρ_{FLOPs}	ρ_{storage}
				$\log L$	E_{train}	E_{test}	$\log L$	E_{train}	E_{test}		
ResNet-20	**R**	0.26	40.55	-0.80	0.22	8.35				1.00	1.00
	1	0.18	26.38	-1.24	0.17	8.87	-1.28	0.05	8.76	1.54	1.54
	2	0.15	22.63	-1.18	0.47	9.08	-1.25	0.13	9.22	1.79	1.78
	4	0.11	17.38	-0.98	1.87	9.72	-1.12	0.79	9.49	2.33	2.36
	8	0.09	13.47	-0.78	4.39	9.91	-0.92	2.51	9.87	3.01	3.16
	16	0.06	10.58	-0.58	8.47	12.26	-0.73	5.18	10.97	3.83	4.18
	32	0.05	8.94	-0.41	13.89	14.67	-0.58	7.95	12.89	4.54	5.17
ResNet-32	**R**	0.46	68.86	-0.82	0.06	7.14				1.00	1.00
	1	0.28	40.17	-1.36	0.02	7.98	-1.38	0.00	8.03	1.71	1.68
	2	0.24	35.07	-1.33	0.07	8.26	-1.36	0.01	8.27	1.96	1.91
	4	0.18	26.55	-1.21	0.47	8.42	-1.30	0.11	8.37	2.59	2.63
	8	0.13	20.18	-0.96	2.44	9.50	-1.13	0.89	9.25	3.41	3.60
	16	0.09	14.79	-0.65	7.28	10.96	-0.83	3.77	10.56	4.66	5.23
	32	0.07	12.00	-0.42	12.08	14.38	-0.65	6.50	11.52	5.74	6.74
ResNet-56	**R**	0.85	125.49	-0.81	0.02	6.58				1.00	1.00
	1	0.45	63.35	-1.41	0.01	7.16	-1.43	0.00	7.18	1.98	1.89
	2	0.41	57.61	-1.41	0.01	7.17	-1.43	0.00	7.35	2.18	2.08
	4	0.29	43.12	-1.37	0.07	7.74	-1.40	0.01	7.52	2.91	2.93
	8	0.20	30.38	-1.16	1.02	8.38	-1.30	0.22	8.34	4.13	4.28
	16	0.13	20.98	-0.78	4.90	9.80	-1.01	1.89	9.08	5.98	6.49
	32	0.10	16.01	-0.53	9.92	12.69	-0.75	4.96	11.13	7.84	8.43
ResNet-110	**R**	1.7	252.88	-0.77	0.01	6.02				1.00	1.00
	1	0.79	100.53	-1.41	0.01	6.75	-1.42	0.00	6.73	2.52	2.20
	2	0.72	93.76	-1.43	0.00	6.83	-1.44	0.00	6.65	2.70	2.40
	4	0.50	71.78	-1.41	0.02	6.98	-1.42	0.01	6.72	3.52	3.44
	8	0.33	50.64	-1.32	0.21	7.56	-1.37	0.05	7.27	4.99	5.18
	16	0.22	33.75	-0.96	2.49	9.55	-1.15	0.89	8.75	7.49	7.81
	32	0.16	24.89	-0.53	7.32	12.60	-0.85	3.50	9.85	10.16	10.72

Table 4: Detailed table of our experiments on ResNet20, 32, 56, 110. We report: training loss L and training and test classification error E_{train} and E_{test} (%); reduction of FLOPs ρ_{FLOPs} and reduction of parameters ρ_{storage}; logarithms are base 10.

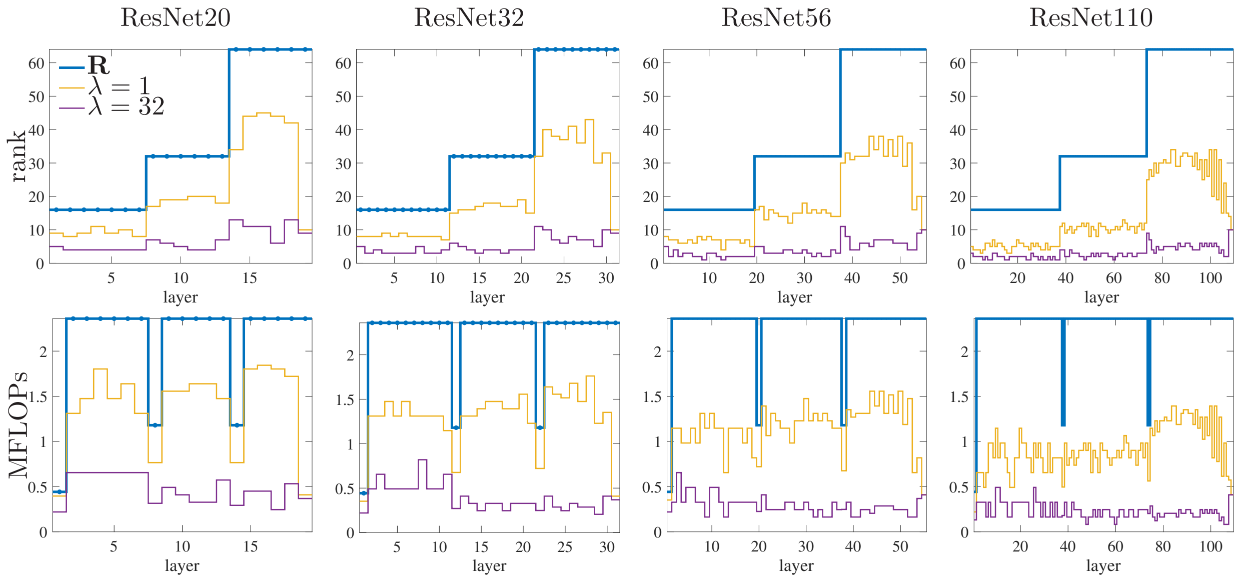

Figure 7: Some final selected architectures in terms of selected rank and resulting FLOPs per layer, when using our method for different λ values on ResNet-s. The multiplicative factor of $\times 10^{-3}$ of λ is omitted. The ranks of last fully-connected layer are not depicted, as the layer was not the part of the rank-selection process. There is a couple of interesting observation we can make. First, the rank of the pen-ultimate layer is always near the value of 10, no matter whether we are enforcing a lot of compression ($\lambda = 32 \times 10^{-3}$) or not much ($\lambda = 1 \times 10^{-3}$). Second, the selected architectures per network, relate in a non-linear way to each other, and a simple rule-based approach will not work.

4.4 Network-in-Network (NIN) on CIFAR10

Network-in-Network (NIN) is a neural network architecture proposed by Lin et al. [14]. Although being widely known and cited, the exact architecture and the training procedure are not given in the original paper[2]. Therefore we referred to a paper of Tai et al. [19], and their code in Torch[3] to train the network for CIFAR10 dataset.

Reference network contains 9 layers and 965568 parameters, and fully outlined in Table 5. The weights were initialized following He et al. [3]. The network was trained using the cross entropy loss with ℓ_2 weight decay of 5×10^{-4}, using Nesterov's accelerated SGD with an initial learning rate of 0.1 for 900 epochs. The learning rate was scaled by 0.1 at epochs 350, 600, 700, 800. During the training we used a simple augmentation consisting of random horizontal flip, zero padding with 4 pixels on each side followed by random crop of 32×32. The resulting test accuracy was 8.87%, which is similar to the result of 8.81% described in [14].

Rank selection experiments are run for 60 LC iterations, with $\mu = 2 \cdot 10^{-5} \times 1.2^k$ at k-th iteration. Each L-step is performed by Nesterov's SGD with momentum 0.9 and run for 20 epochs with learning rate of 0.0007 at the beginning of the step and decayed by 0.99 after each epoch. Fine-tuning is performed on the network which directly parametrize low rank as a sequence of fully-connected or convolutional layers with Nesterov's SGD with momentum 0.9 for 500 epochs, with the initial learning rate 5×10^{-4} and decayed by 0.99 after each epoch. *Runtime.* Running the LC with fine-tuning is 3.5 times longer comparing to the training of the reference network.

Baselines. As discussed in Section 3, we train models with two different baselines. We remind that *baseline 1* is due to Zhang et al. [24], and *baseline 2* is a simple singular value thresholding scheme used across the literature. Every baseline is controlled by a single hyperparameter, by varying which we obtain different low-rank neural networks. Each obtained network is fine-tuned using Nesterov's SGD with learning rate of 0.001 (decayed by 0.99 after every epoch) and momentum of 0.9 for 600 epochs. The comparison between our algorithm and the baselines is given in Figure 8.

Layer	Connectivity
Input	32×32 image
1	convolutional, 192 5×5 filters (stride=1, padding=2), followed by BN and ReLU
2	convolutional, 160 1×1 filters (stride=1, padding=0), followed by BN and ReLU
3	convolutional, 96 1×1 filters (stride=1, padding=0), followed by BN and ReLU
	max pool, 3×3 window (stride=2, padding=1)
4	convolutional, 192 5×5 filters (stride=1, padding=2), followed by BN and ReLU
5	convolutional, 192 1×1 filters (stride=1, padding=0), followed by BN and ReLU
6	convolutional, 192 1×1 filters (stride=1, padding=0), followed by BN and ReLU
	max pool, 3×3 window (stride=2, padding=1)
7	convolutional, 192 3×3 filters (stride=1, padding=1), followed by BN and ReLU
	max pool, 2×2 window (stride=2)
8	convolutional, 192 1×1 filters (stride=1, padding=0), followed by BN and ReLU
9	convolutional, 10 1×1 filters (stride=1, padding=0), followed by BN and ReLU
(output)	max pool, 3×3 window (stride=2, padding=1), followed by softmax
	965568 weights, 1418 biases, 1418 running means/variances for BN

Table 5: Structure of the adapted NIN network for the CIFAR10 dataset. BN–Batch Normalization, ReLU – rectified linear units.

[2] However, authors do provide a link to config of the network, in https://github.com/mavenlin/cuda-convnet/

[3] https://github.com/chengtaipu/lowrankcnn

	$\lambda \times 10^{-5}$	# params $\times 10^6$	MFLOPs	Before fine-tuning			After fine-tuning			ρ_{FLOPs}	ρ_{storage}
				$\log L$	E_{train}	E_{test}	$\log L$	E_{train}	E_{test}		
NIN	**R**	0.97	222.48	-0.77	0.16	8.87					
	2.5	0.47	106.85	-1.41	0.17	9.24	-1.47	0.05	8.86	2.08	2.05
	5	0.42	90.98	-1.43	0.38	9.61	-1.55	0.13	9.38	2.45	2.30
	7.5	0.36	80.43	-1.35	0.73	9.80	-1.50	0.31	9.53	2.77	2.71
	10	0.32	73.25	-1.27	1.03	10.19	-1.42	0.54	9.79	3.04	3.03
	15	0.27	64.22	-1.12	2.01	10.38	-1.28	1.10	9.72	3.46	3.54
	20	0.25	58.58	-1.04	2.63	10.46	-1.18	1.65	10.11	3.80	3.90
	25	0.23	53.60	-0.94	3.41	11.00	-1.08	2.31	10.69	4.15	4.28
	30	0.22	51.30	-0.89	3.98	10.84	-1.03	2.74	10.45	4.34	4.46
	40	0.20	46.10	-0.78	5.16	11.61	-0.92	3.61	10.93	4.83	4.96
	50	0.18	42.85	-0.72	6.09	12.05	-0.85	4.44	11.54	5.19	5.25
	60	0.17	40.35	-0.65	7.41	12.51	-0.79	5.33	11.56	5.51	5.54
	70	0.16	37.44	-0.57	9.00	14.36	-0.72	6.25	12.55	5.94	5.95
	90	0.14	33.57	-0.44	12.50	17.03	-0.63	7.78	13.08	6.63	6.69
	130	0.13	29.67	-0.33	15.75	18.69	-0.53	9.83	14.71	7.50	7.49
	140	0.13	28.82	-0.31	16.47	19.09	-0.50	10.57	15.17	7.72	7.63
	200	0.12	27.07	-0.27	18.24	20.60	-0.46	11.77	15.44	8.22	8.02
	300	0.11	23.67	0.04	35.99	37.45	-0.41	13.36	17.44	9.40	9.01
	350	0.10	22.53	0.01	34.24	35.63	-0.28	18.18	21.27	9.88	9.76

Table 6: Detailed table of our experiments on the NIN. We report: training loss L and training and test classification error E_{train} and E_{test} (%); reduction of FLOPs ρ_{FLOPs} and reduction of parameters ρ_{storage}; logarithms are base 10.

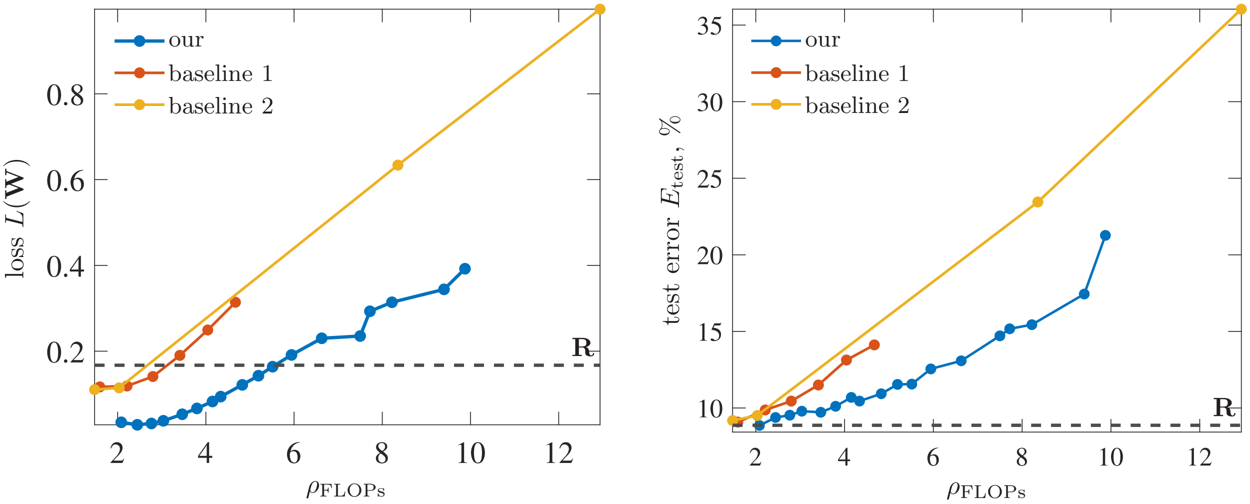

Figure 8: Comparison of our rank selection algorithm to the baselines on the Network-in-Network (NIN). *Baseline 1* is due to Zhang et al. [24], and *baseline 2* is a simple singular value thresholding scheme used across the literature. Left figure: loss vs FLOPs reduction ratio (ρ_{FLOPs}), right figure: test error vs ρ_{FLOPs}. Our algorithm achieves both lower loss and better accuracy throughout. For details of training refer to the text.

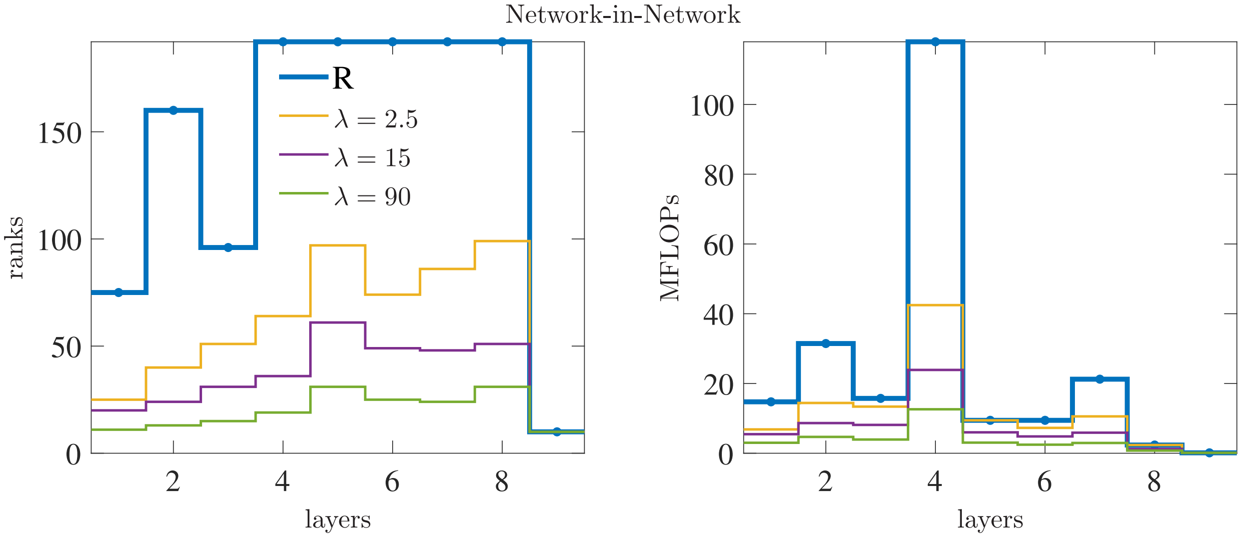

Figure 9: Some selected architectures for our algorithm on NIN. For λ values, the multiplicative factor of $\times 10^{-5}$ is omitted.

4.5 AlexNet on ImageNet

AlexNet [9] is one of the first convolutional networks trained on large scale image recognition task — ImageNet [17]. The dataset contains 1.2M colored images of various resolution of 1000 different classes. For all AlexNet experiments we report single view top-1 and top-5 errors (see below for more details).

Reference Network We train Batch Normalized version of the AlexNet having 62M parameters (see Table 7), closely following the data-augmentation scheme of the original paper [9]. For completeness, we give the data-augmentation scheme here: images are resized to have the smallest dimension of 256 pixels, and random 227×227 image is cropped during the training. Cropped images are normalized, and lightening augmentation of [9] is applied. During testing, a central crop is obtained. We train reference model for 100 epochs with an initial learning rate of 0.05 and weight decay amount of 5×10^{-5}. We decrease the learning rate by factor of 0.1 every 20 epochs. The reference model achieves Top-1 validation accuracy of 57.71% and Top-5 validation accuracy of 80.45%. *Training time* on Titan V GPU is 17 hours.

Layer	Connectivity
Input	227×227 image
1	convolutional, 96 11×11 filters (stride=4, padding=2) $\rightarrow$ BN $\rightarrow$ ReLU max pool, 3×3 window (stride=2, padding=0)
2	convolutional, 256 5×5 filters (stride=1, padding=2) $\rightarrow$ BN $\rightarrow$ ReLU max pool, 3×3 window (stride=2, padding=0)
3	convolutional, 384 3×3 filters (stride=1, padding=1) $\rightarrow$ BN $\rightarrow$ ReLU max pool, 3×3 window (stride=2, padding=1)
4	convolutional, 384 3×3 filters (stride=1, padding=1) $\rightarrow$ BN $\rightarrow$ ReLU max pool, 3×3 window (stride=2, padding=0)
5	convolutional, 256 3×3 filters (stride=1, padding=1) $\rightarrow$ BN $\rightarrow$ ReLU max pool, 3×3 window (stride=2, padding=0)
6	fully connected, 9216×4096 $\rightarrow$ BN $\rightarrow$ ReLU $\rightarrow$ Dropout
7	fully connected, 4096×4096 $\rightarrow$ BN $\rightarrow$ ReLU $\rightarrow$ Dropout
8	fully connected, 4096×1000
(output)	softmax
62 378 344 weights, 1000 biases, 9568 running means/variances for BN, 1139 MFLOPs	

Table 7: The structure of BatchNormalized AlexNet network for the ImageNet dataset. BN–Batch Normalization, ReLU – rectified linear units.

Our BN-AlexNet has slightly more floating point operations comparing to the original paper, 1139M vs. 724M, which is due to the absence of group convolutions on the layers 1,3, and 5. To avoid confusion, whenever we are comparing to original AlexNet or Caffe-AlexNet (as reported in [2]), we will be reporting achieved number of FLOPs in the final model.

Rank Selection experiments We run our experiments for 30 LC steps with $\mu_k = 0.5 \times 10^{-3} \times 1.2^k$ at k-th L-step. All L-steps are run for 5 epoch with a minibatch size of 256, the first 20 L-steps have initial learning rate of 0.001 decayed by 0.9 at every epoch, and the last 10 L-steps have initial learning rate 0.0005 with decay factor of 0.9 after every epoch. Learning rates for each L-step are restarted. We finally fine-tune models for 10 epochs using learning rate of 0.0005, decayed by 0.9 after every epoch, and minibatch size of 512. *Runtime:* 30 LC steps with the final fine-tuning finishes in 32 hours on NVIDIA Titan V GPU.

	$\lambda \times 10^{-4}$	# params,	MFLOPs	E_{val}, %		ρ_{FLOPs}	ρ_{storage}
		$\times 10^6$		top-1	top-5		
	R	62.3	1139	42.29	19.54	1.00	1.00
scheme 1	0.05	40.7	436	41.56	19.15	2.61	1.53
	0.15	18.2	263	42.63	19.95	4.33	3.43
	0.17	14.2	240	42.83	19.93	4.75	4.39
scheme 2	0.05	40.5	324	41.46	19.14	3.52	1.53
	0.10	25.2	236	41.81	19.40	4.83	2.47
	0.15	18.1	190	42.07	19.54	5.99	3.45
	0.20	12.4	151	42.69	19.83	7.57	5.01

Table 8: Rank Selection on AlexNet with different λ values using the low-rank parametrization schemes 1 and 2 (after fine-tuning). We report: top-1/5 errors on the validation set E_{val}; resulting number of MFLOPs and parameters; FLOPs reduction ratio ρ_{FLOPs} and storage reduction ratio ρ_{storage}.

	MFLOPs	top-1	top-5	ρ_{FLOPs}
Caffe-AlexNet [6]	724	42.70	**19.80**	1.00
Kim et al. [8], Tucker	272	n/a	21.67	2.66
Tai et al. [19], scheme 2	185	n/a	20.34	3.90
Wen et al. [20], scheme 1	269	n/a	20.14	2.69
Kim et al. [7], scheme 2	272	43.40	20.10	2.66
Yu et al. [23], filter pruning	232	44.13	n/a	3.12
Li et al. [13], filter pruning	334	43.17	n/a	2.16
Ding et al. [1], filter pruning	492	43.83	20.47	1.47
our, scheme 1, $\lambda = 0.17$	240	42.83	19.93	3.01
our, scheme 2, $\lambda = 0.20$	**151**	**42.69**	19.83	**4.79**

Table 9: Comparison of our low-rank AlexNet-s to results in literature obtained by decomposition methods (including low-rank) and structured pruning. Reduction of FLOPs (ρ_{FLOPs}) is given wrt Caffe-AlexNet, we can achieve 151MFLOPs (×4.79 reduction wrt Caffe-AlexNet) with better error.

4.6 Discussion and comparison

In this section we provide a comprehensive evaluation of our algorithm to the best results in the literature on speeding up the neural networks on the CIFAR10 dataset (for the ImageNet comparison, please refer to section 4.5). We have to note that most of the approaches in literature relies on structured pruning, with only a small fraction being low-rank networks. We will demonstrate that our approach does not only outperform other low-rank approaches but generally competitive with pruning.

Our algorithm vs baselines To understand why our algorithm achieves better results, we plot selected ranks and FLOPs for us, and two baselines, for the final architectures that have about the same FLOPs reduction on NIN. Our network achieves ×2.08, baseline 1 — ×2.21, and baseline 2 — ×2.03 reduction in FLOPs while in terms of accuracy the results are 91.14%, 90.13%, and 90.49% respectively (Figure 10).

With multiple quantities of interest comparing the performance of compressed neural networks is rather tricky. The most obvious way is to report a single compression ratio in terms of the number of parameters, or speed-up. Having only one number does not necessarily reflect other important metrics, e.g., compression of the parameters does not correspond to a faster inference (with fewer FLOPs), and generally, not as interesting as the interplay between compressed model's performance, compression and speed-up ratios. We should also note that the compression ratios (of any kind) on its own are not representative as they can be easily inflated by compressing a larger (overparametrized) model in the first place. Therefore, to visualize and understand this tradeoff better, we decided to report achieved FLOPs, model size and test accuracy in single Fig. 13.

Fig. 13 depicts all our CIFAR10 results obtained via low-rank compression (as connected circles), other's results obtained via low-rank compression as labeled circles [11, 20, 21], and most importantly puts low-rank compression in perspective with other reported results for faster inference, i.e., structured filter pruning [5, 12, 22, 23, 25], as squares (to indicate apples to oranges comparison). Ideally, we would like to have models on the left-bottom corner of this plot, where both FLOPs and error are minimal. Results trace a pareto curve, which is *mostly formed by our low-rank* compressed ResNet-s and VGG16. We make a few observations: 1) low-rank models obtained via our algorithm are *comparable and often considerably better*

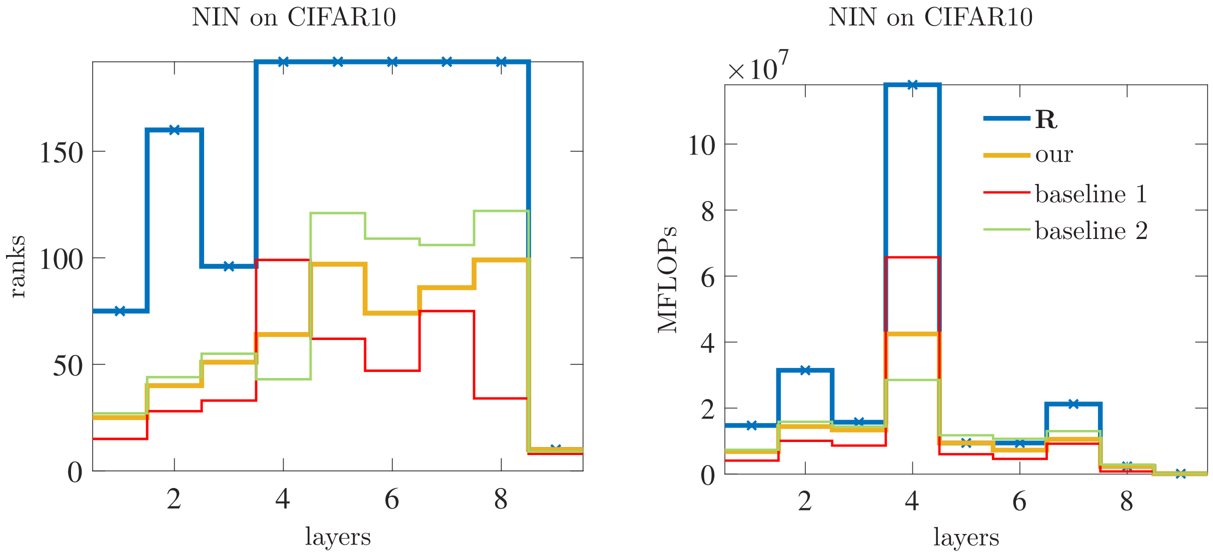

Figure 10: Comparison of selected ranks and corresponding FLOPs achieved by our algorithm and the baselines on NIN-CIFAR10. The resulting FLOPs reduction for our method—×2.08, baseline 1 — ×2.21, baseline 2—×2.03; with corresponding accuracies of 91.14%, 90.13%, and 90.49%. As you can see, for the same amount of reduction in FLOPs, our results has more than 0.65% of margin. More interestingly, you can see that most of the selected ranks of both baselines do not agree with our selected ranks, and the only agreeing one is in the last 9th layer.

than other low-rank compression and structured pruning results 2) it is often beneficial in terms of error-FLOPs tradeoff to train a larger model and then compress it, e.g., one of the low-rank VGG16-s achieve 6.11% error with 107 MFLOPs, while reference ResNet110 has 6.02% error with 252 MFLOPs.

In Fig. 11, Fig. 12, and Fig. 14 we produce a single plot of FLOPs vs test accuracy for each of the VGG, ResNet and NIN networks. The size of every circle is proportional to the achieved number of parameters. In these plots, the most ideal networks are small circles located in the left bottom position. In order to put our results in perspective, we plot alongside our results, the results from literature. If there are existing methods that use low-rank decomposition we plot them using circles, and everything else as squares. (To illustrate apples to oranges comparison). We can see that low rank models achieved using our method outperforms all other low-rank based methods reported in literature and competitive with structured pruning methods, outperforming them on certain occasions.

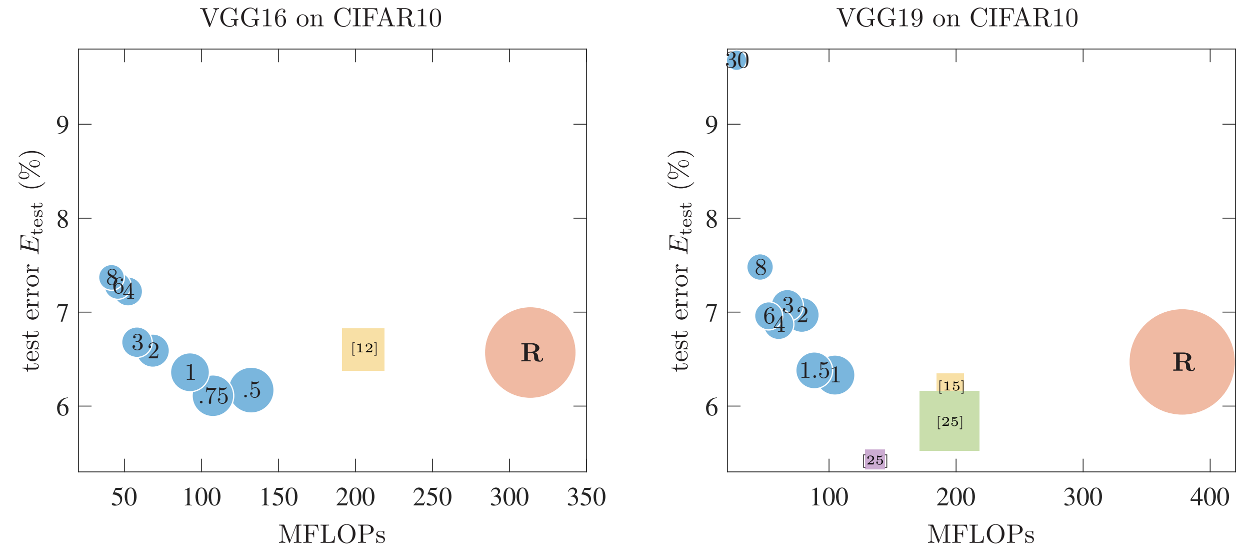

Figure 11: Depiction of an interplay between model FLOPs, number of parameters, λ values and resulting test error for LC Models, as in Table 3. Each blue circle corresponds to a particular obtained model; the number of its parameters is the area of the circle; the label inside is the value of λ it was trained with (the multiplicative factor of $\times 10^{-4}$ omitted). We show the interplay for VGG16 (on the left) and VGG19 (on the right) trained on CIFAR10, **R** – the reference model. Other colored circles are the comparison points from the literature, in this case, obtained via filter pruning.

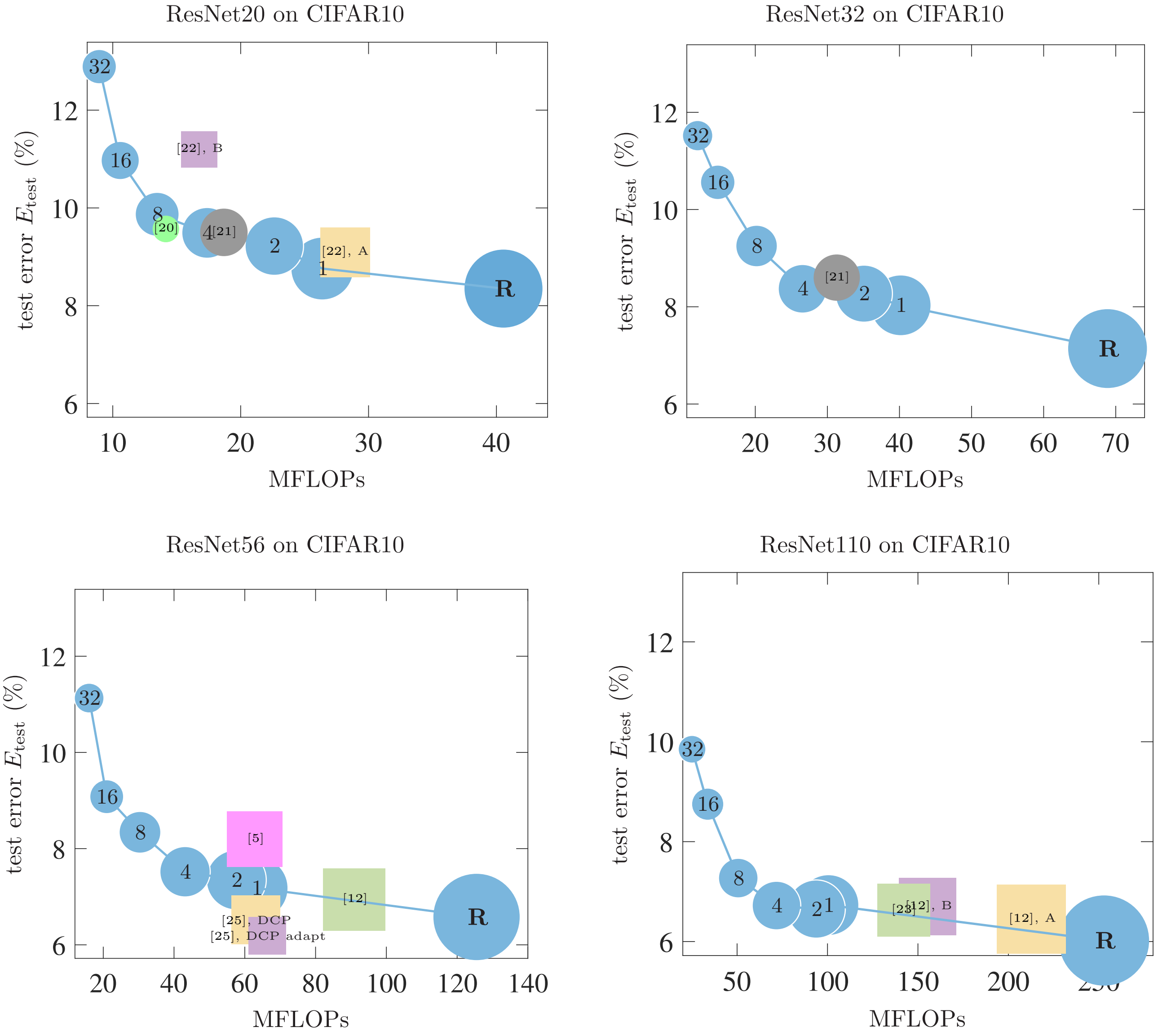

Figure 12: Depiction of an interplay between model FLOPs, number of parameters, λ values and resulting test error for LC Models, as in Table 4. Each blue circle corresponds to a particular compressed model via LC; the number of its parameters is the area of the circle; the label inside is the value of λ it was trained with (the multiplicative factor of $\times 10^{-3}$ omitted). For comparison, we show results of other compression methods via circles if they are using low rank, and via squares if it is something else (e.g., pruning).

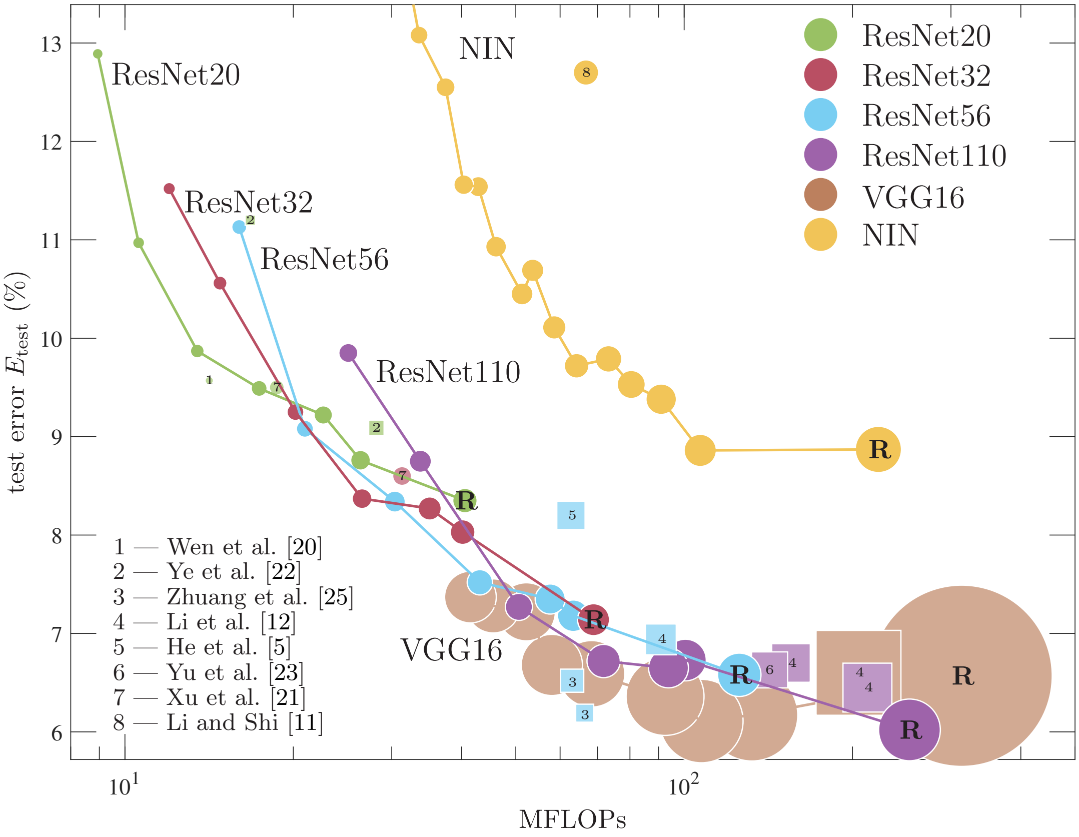

Figure 13: Error-compression space of test error (Y axis), inference FLOPs (X axis) and number of parameters (ball size for each net), for ResNets, VGG16 and NIN trained on CIFAR-10. Results of our algorithm over different λ values for a given net span a curve, shown as connected circles ●—●, which starts on the lower right at the reference **R** ($\lambda = 0$) and then moves left and up. Other published results using low-rank compression are shown as isolated circles labeled with a citation. Other published results involving structured filter pruning for faster inference are shown as isolated squares labeled with a citation. Each color corresponds to a different reference net. The area of a circle or square is proportional to the number of parameters in the corresponding compressed model. Ideal models are small balls on the left-bottom, where both error and FLOPs are the smallest.

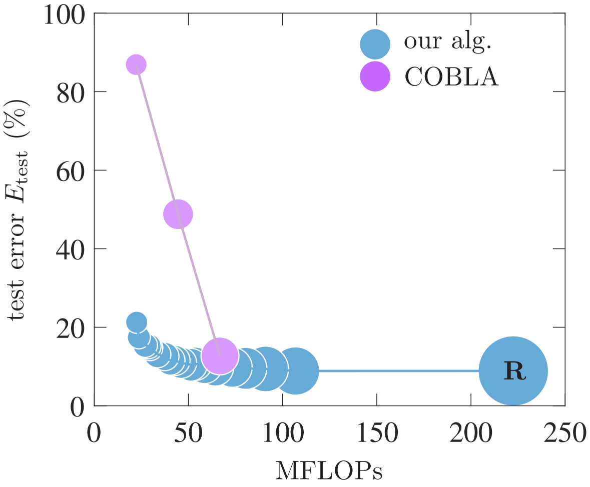

Figure 14: Depiction of an interplay between model FLOPs, number of parameters, λ values and resulting test error for NIN models. Each blue circle correspond to a particular obtained model via LC, the number of its parameters is the area of the circle, label inside is the value of λ it was trained with (the multiplicative factor of $\times 10^{-4}$ omitted). **R** – the reference model. Pink colored circles are the results of training NIN with low-rank decompositions using COBLA by [11].

network	err	# params	ρ_{storage}	MFLOPs	ρ_{FLOPs}
R ResNet20	8.35%	269 722	1.00	40.55	1.00
pruning, [22], A	9.09%	176 596	1.52	28.16	1.44
pruning, [22], B	11.20%	89 344	3.01	16.75	2.42
low-rank, [20]	9.57%	31 352	8.60	14.15	2.87
low-rank, [21]	9.50%	100 000	2.69	18.68	2.17
our, $\lambda = 1 \cdot 10^{-3}$	8.76%	175 137	1.54	26.38	1.54
our, $\lambda = 8 \cdot 10^{-3}$	9.87%	85 351	3.16	13.47	3.01
R ResNet32	7.14%	464 144	1.00	68.86	1.00
low-rank, [21]	8.60%	160 000	2.90	31.30	2.20
our, $\lambda = 4 \cdot 10^{-3}$	8.37%	176 480	2.63	26.55	2.59
R ResNet56	6.58%	853 008	1.00	125.49	1.00
pruning, [12]	6.94%	736 145	1.15	90.85	1.38
pruning, [5]	8.20%	590 143	1.45	62.74	2.00
DCP pruning, [25]	6.51%	432 998	1.97	63.05	1.99
DCP-Adapt pruning, [25]	6.19%	253 118	3.97	66.39	1.89
our, $\lambda = 1 \cdot 10^{-3}$	7.18%	451 326	1.89	63.35	1.98
R ResNet110	6.02%	1 727 952	1.00	252.88	1.00
pruning, [12], A	6.45%	1 688 209	1.15	212.67	1.18
pruning, [12], B	6.70%	1 168 095	1.15	155.27	1.62
pruning, [23]	6.61%	980 612	1.97	142.17	1.77
our, $\lambda = 4 \cdot 10^{-3}$	6.72%	502 311	3.44	71.78	3.52

Table 10: Comparison of results based on low-rank architecture learned using our method and others on ResNets.

network	err	# params	ρ_{storage}	MFLOPs	ρ_{FLOPs}
R VGG16	6.57%	14.9M	1.00	313	1.00
our, $\lambda = 2 \cdot 10^{-4}$	6.59%	2.0M	7.48	68	4.59
filter pruning, [12]	6.60%	5.4M	2.77	207	1.51
R VGG19	6.46%	20.2M	1.00	398	1.00
our, $\lambda = 1.5 \cdot 10^{-4}$	6.38%	2.47M	8.21	88	4.51
filter pruning, [15]	6.20%	2.30M	8.69	196	2.03
DCP-adapt, [25]	5.43%	1.29M	15.58	140	2.86
DCP, [25]	5.84%	10.5M	1.92	195	2.00

Table 11: Comparison of results of low-rank architecture learned using our method on VGG-s and the recent VGG architectures obtained via filter pruning on the CIFAR10.

References

[1] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan. Approximated oracle filter pruning for destructive CNN width optimization. In K. Chaudhuri and R. Salakhutdinov, editors, *Proc. of the 36th Int. Conf. Machine Learning (ICML 2019)*, pages 1607–1616, Long Beach, CA, June 9–15 2019.

[2] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proc. 15th Int. Conf. Computer Vision (ICCV'15)*, pages 1026–1034, Santiago, Chile, Dec. 11–18 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, Las Vegas, NV, June 26 – July 1 2016.

[5] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proc. 17th Int. Conf. Computer Vision (ICCV'17)*, pages 1398–1406, Venice, Italy, Dec. 11–18 2017.

[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093 [cs.CV], June 20 2014.

[7] H. Kim, M. U. K. Khan, and C.-M. Kyung. Efficient neural network compression. In *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, pages 12569–12577, Long Beach, CA, June 16–20 2019.

[8] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016.

[9] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 25, pages 1106–1114. MIT Press, Cambridge, MA, 2012.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov. 1998.

[11] C. Li and C. J. R. Shi. Constrained optimization based low-rank approximation of deep neural networks. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Proc. 15th European Conf. Computer Vision (ECCV'18)*, pages 746–761, Munich, Germany, Sept. 8–14 2018.

[12] H. Li, A. Kadav, I. Durdanovic, and H. P. Graf. Pruning filters for efficient ConvNets. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 24–26 2017.

[13] J. Li, Q. Qi, J. Wang, C. Ge, Y. Li, Z. Yue, and H. Sun. OICSR: Out-in-channel sparsity regularization for compact deep neural networks. In *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, pages 7046–7055, Long Beach, CA, June 16–20 2019.

[14] M. Lin, Q. Chen, and S. Yan. Network in network. In *Int. Conf. Learning Representations (ICLR 2014)*, 2014.

[15] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *Proc. 17th Int. Conf. Computer Vision (ICCV'17)*, Venice, Italy, Dec. 11–18 2017.

[16] Y. Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Math. Dokl.*, 27(2):372–376, 1983.

[17] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *Int. J. Computer Vision*, 115(3):211–252, Dec. 2015.

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015.

[19] C. Tai, T. Xiao, Y. Zhang, X. Wang, and W. E. Convolutional neural networks with low-rank regularization. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016.

[20] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li. Coordinating filters for faster deep neural networks. In *Proc. 17th Int. Conf. Computer Vision (ICCV'17)*, Venice, Italy, Dec. 11–18 2017.

[21] Y. Xu, Y. Li, S. Zhang, W. Wen, B. Wang, Y. Qi, Y. Chen, W. Lin, and H. Xiong. Trained rank pruning for efficient deep neural networks. arXiv:1812.02402, Dec. 8 2018.

[22] J. Ye, X. Lu, Z. Lin, and J. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *Proc. of the 6th Int. Conf. Learning Representations (ICLR 2018)*, Vancouver, Canada, Apr. 30 – May 3 2018.

[23] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. NISP: Pruning networks using neuron importance score propagation. In *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, pages 9194–9203, Salt Lake City, UT, June 18–22 2018.

[24] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, Oct. 2016.

[25] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pages 815–886. MIT Press, Cambridge, MA, 2018.