Technical Appendix of "1-Lipschitz Layers Compared: Memory, Speed, and Certifiable Robustness"

A. Spectral norm and orthogonalization

A lot of recently proposed methods do rely on a way of parameterizing orthogonal matrices or parameterizing matrices with bounded spectral norm. We present methods that are frequently used below:

Bjorck & Bowie [5] introduced an iterative algorithm that finds the closest orthogonal matrix to the given input matrix. In the commonly used form, this is achieved by computing a sequence of matrices using

$$A_{k+1} = A_k \left(I + \frac{1}{2} Q_k \right), \quad \text{for } Q_k = I - A_k^\top A_k \tag{9}$$

where $A_0 = A$, is the input matrix. The algorithm is usually truncated after a fixed number of steps, during training often 3 iterations are enough, and for inference more (e.g. 15) iterations are used to ensure a good approximation. Since the algorithm is differentiable, it can be applied to construct 1-Lipschitz networks as proposed initially in [2] or also as an auxiliary method for more complex strategies [27].

Power Method The *power method* was used in [32], [24] and [31] in order to bound the spectral norm of matrices.
It starts with a random initialized vector $\mathbf{u}_0$, and iteratively applies the following:

$$\mathbf{v}_{k+1} = \frac{W^\top \mathbf{u}_k}{\|W^\top \mathbf{u}_k\|_2}, \quad \mathbf{u}_{k+1} = \frac{W \mathbf{v}_{k+1}}{\|W \mathbf{v}_{k+1}\|_2}. \tag{10}$$

Then the sequence σ_k converges to the spectral norm of W, for σ_k given by

$$\sigma_k = \mathbf{u}_k^\top W \mathbf{v}_k. \tag{11}$$

This procedure allows us to obtain the spectral norm of matrices, but it can also be efficiently extended to find the spectral norm of the Jacobian of convolutional layers. This was done for example by [13, 24], using the fact that the transpose of a convolution operation (required to calculate Equation (10)) is a convolution as well, with a kernel that can be constructed from the original one by transposing the channel dimensions and flipping the spatial dimensions of the kernel.

When the power method is used on a parameter matrix of a layer, we can make it even more efficient with a simple trick. We usually expect the parameter matrix to change only slightly during each training step, so we can store the result $\mathbf{u}_k$ during each training step, and start the power method with this vector as $\mathbf{u}_0$ during the following training step. With this trick it is enough to do a single iteration of the power method at each training step. The power method is usually not differentiated through.

Fantasic Four proposed, in [35], allows upper bounding the Lipschitz constant of a convolution. The given bound is generally not tight, so using the method directly does not give good results. Nevertheless, since various methods require a way of bounding the spectral norm to have convergence guarantees, Fantastic Four is often used.

B. Algorithms omitted in the main paper

Observe that the strategies presented in [9, 19, 25, 28, 32, 42, 47] have intentionally not been compared for different reasons. In the works presented in [9, 28], the Lipschitz constraint was solely used during training and no guarantees were provided that the resulting layers are 1-Lipschitz. The method proposed in [32] has been extended by Fantastic 4 [35] and, indeed, can only be used as an auxiliary method to upper-bound the Lispchitz constant. The method proposed in [25] only works for linear layers and can be thought of as a special case of SOC (described in Section 2). We will give detailed reasons for the other methods below.

ONI The method ONI [19] proposed the orthogonalization used in LOT. They parameterize orthogonal matrices as $(VV^\top)^{-\frac{1}{2}}V$, and calculate the inverse square root using Newton's iterations. They use this methods to define 1-Lipschitz linear layers. However, the extension to convolutions only uses a simple unrolling, and does not provide a tight bound in general. Therefore, we did not include the method in the paper.

ECO *Explicitly constructed orthogonal (ECO)* convolutions [47] also do use properties of the Fourier domain in order to parameterize a convolution. However, they do not actually calculate the convolution in the Fourier domain, but instead parameterize a layer in the Fourier domain, and then use an inverse Fourier transformation to obtain a kernel from this parameterization. We noticed, however, that the implementation provided by the authors does not produce 1-Lipschitz layers (at least with our architecture), as can be seen in Figure 5. There, we report the *batch activation variance* (defined in Appendix F) as well as the spectral norm of each layer. The *batch activation variance* should be non-increasing for 1-Lipschitz layers (also see Appendix F), however, for ECO this is not the case. Also, power iteration shows that the Lipschitz constant of individual layers is not 1. Therefore we do not report this method in the main paper.

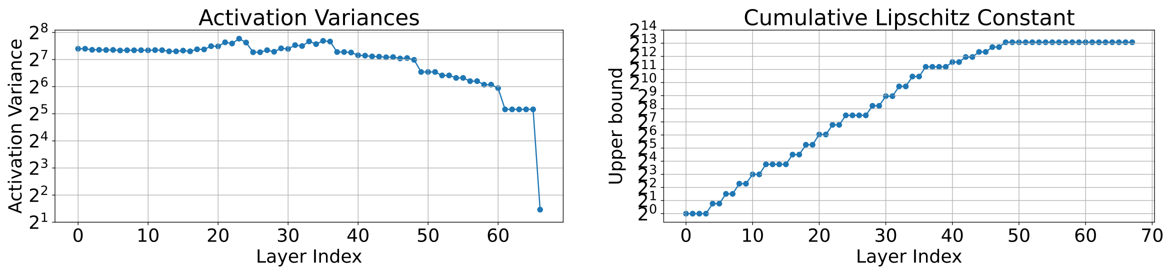

Figure 5. **Left:** Variance of a validation batch over the batch dimension. For 1-Lipschitz layers, this property should be non-increasing, as proven in Appendix F. **Right:** Upper bound on the Lipschitz Constant applying the power method to every linear layer, and multiplying the results. Plot for ECO on CIFAR-10, with model S.

Sandwich The authors of [42] introduced the *Sandwich* layer. It considers a layer of the form

$$l(x) = \sqrt{2}A^T \Psi \mathrm{ReLU}\left(\sqrt{2}\Psi^{-1}Bx + b\right), \tag{12}$$

for σ typically the ReLU activation. The authors propose a (simultaneous) parameterization of A and B, based on the Cayley Transform, that guarantees the whole layer to be 1-Lipschitz. They also extend the idea to convolutions. However, for this they require to apply two Fourier transformations as well as two inverse ones. During the training of the models within the Sandwich layers, a severe vanishing gradient phenomena happens. We summarize the Frobenious norm of the gradient, obtained by inspecting the inner blocks during the training, in Table 3. For this reason we did not report the results in the main paper.

C. Computation Complexity and Memory Requirement

In this section we give some intuition of the values in Table 1. Recall that we consider a layer with input size $s \times s \times c$, and kernel size $k \times k$, batch size b, and (for some layers) we will denote the number of inner iterations by t. We also use $C = bs^2c^2k^2$ and $M = bs^2c$ and $P = c^2k^2$.

AOL: In order to compute the rescaling matrix for AOL, we need to convolve the kernel with itself. This operation has complexity $\mathcal{O}(c^3k^4)$. It outputs a tensor of size $c \times c \times (2k-1) \times (2k-1)$, so in total we require memory of about $5P$ for the parameter as well as the transformation.

BCOP: For BCOP we only require a single convolution as long as we know the kernel. However, we do require a lot of computation to create the kernel for this convolution. In particular, we require $2k-1$ matrix orthogonalizations (usually done with Bjorck & Bowie), as well as $\mathcal{O}(k^3)$ matrix multiplications for building up the kernel. These require about $c^3kt + c^3k^3$ MACS as well as $c^2kt + c^2k^3$ memory.

Cayley: Cayley Convolutions make use of the fact that circular padded convolutions are vector-matrix products in the Fourier domain. Applying the fast Fourier transform to inputs and weights has complexity of $\mathcal{O}(bcs^2 \log(s^2))$ and $\mathcal{O}(c^2 s^2 \log(s^2))$. Then, we need to orthogonalize $\frac{1}{2}s^2$ matrices. Note that the factor of $\frac{1}{2}$ appears due to the fact that the Fourier transform of a real matrix has certain symmetry properties, and we can use that fact to skip half the computations. Doing the matrix orthogonalization with the Cayley Transform requires taking the inverse of a matrix, as well as matrix multiplication, the whole process has a complexity of about s^2c^3. The final steps consists of doing $\frac{1}{2}bs^2$ matrix-vector products, requiring $\frac{1}{2}bs^2c^2$ MACS, as well as another fast Fourier transform . Note that under our assumption that $c > \log(s^2)$, the fast Fourier transform operation is dominated by other operations. Cayley Convolutions require padding the kernel from a size of $c \times c \times k \times k$ to a (usually much larger) size of $c \times c \times s \times s$ requiring a lot of extra memory. In particular we need to keep the output of the (real) fast Fourier transform, the matrix inversion as well as the matrix multiplication in memory, requiring about $\frac{1}{2}s^2c^2$ memory each.

CPL: CPL applies two convolutions as well as an activation for each layer. They also use the Power Method (on the full convolution), however, its computational cost is dominated by the application of the convolutions.

LOT: Similar to Cayley, LOT performs the convolution in Fourier space. However, instead of using the Cayley transform, they parameterize orthogonal matrices as $V(V^TV)^{-\frac{1}{2}}$. To find the inverse square root, authors relay on an iterative *Newton Method*. In details, let $Y_0 = V^TV$ and $Z_0 = I$, then Y_i defined as

$$Y_{i+1} = \frac{1}{2}Y_i\left(3I - Z_iY_i\right), \quad Z_{i+1} = \frac{1}{2}\left(3I - Z_iY_i\right)Z_i, \tag{13}$$

converges to $(V^TV)^{-\frac{1}{2}}$. Executing this procedure, includes computing $4s^2t$ matrix multiplications, requiring about $4s^2c^3t$ MACS as well as $4s^2c^2t$ memory.

SLL: Similar to CPL, each SLL layer also requires evaluating two convolutions as well as one activation. However, SLL also needs to compute the AOL rescaling, resulting in total computational cost of $2C + \mathcal{O}(c^3k^4)$.

SOC: For each SOC layer we require applying t convolutions. Other required operations (application of Fantastic 4 for an initial bound, as well as parameterizing the kernel such that the Jacobian is skew-symmetric) are cheap in comparison.

C.1. Memory requirements at inference time

The amount of memory required at inference time is much lower than what is needed during training. However, it might still be important, for example when someone is interested in efficient deployment of models. During inference time, we do not need to keep the activations in memory, but only need to store them while we are doing computations. Furthermore, all

Table 3. **Vanishing Gradient Phenomena of Sandwhich Layer**. ConvNetXS model has been tested with a small batch size (32). Training of deeper layers (i.e., layers that are close to the input of the network) is tough due to the almost zero gradients.

Layer name	Output Shape	Gradient Norm
First Conv (1×1 kernel size)	$(32, 16, 32, 32)$	$3{,}36 \cdot 10^{-7}$
Activation	$(32, 16, 32, 32)$	$3{,}36 \cdot 10^{-7}$
Downsize Block(3)	$(32, 32, 16, 16)$	$3{,}79 \cdot 10^{-6}$
Downsize Block(3)	$(32, 64, 8, 8)$	$5{,}56 \cdot 10^{-5}$
Downsize Block(3)	$(32, 128, 4, 4)$	$7{,}83 \cdot 10^{-4}$
Downsize Block(3)	$(32, 256, 2, 2)$	$8{,}15 \cdot 10^{-3}$
Downsize Block(1)	$(32, 512, 1, 1)$	$1{,}04 \cdot 10^{-1}$
Flatten	$(32, 512)$	$1{,}04 \cdot 10^{-1}$
Linear	$(32, 512)$	$1{,}82 \cdot 10^{-1}$
First Channels	$(32, 10)$	$1{,}82 \cdot 10^{-1}$

Table 4. **SOTA from the literature on CIFAR-10** sorted by publication date (from older to newer). Readers can note that there is a clear trend of increasing the model dimension to achieve higher robust accuracy. The last three publications train and use diffusion models for data augmentation.

Method	Std.Acc [%]	Certifiable Accuracy [%] $\varepsilon = \frac{36}{255}$	$\varepsilon = \frac{72}{255}$	$\varepsilon = \frac{108}{255}$	$\varepsilon = 1$	Number of Parameters
BCOP Large [27]	72.1	58.2	-	-	-	2M
Cayley KW-Large [40]	75.3	59.1	-	-	-	2M
SOC LipNet-25 [36]	76.4	61.9	-	-	-	24M
AOL Large [34]	71.6	64.0	56.4	49.0	23.7	136M
LOT LipNet-25 [46]	76.8	64.4	49.8	37.3	-	27M
SOC LipNet-15 + CRC [37]	79.4	67.0	52.6	38.3	-	21M
CPL XL [31, Table1]	78.5	64.4	48.0	33.0	-	236M
SLL X-Large [3]	73.3	65.8	58.4	51.3	27.3	236M
LOT + DDPM [1]	81.4	69.1	-	-	-	5M
Gloro LiResNet + DDPM [18]	82.1	70.1	-	-	-	49M
Cholensky LiResNet + EDM [17]	**87.0**	**78.1**	**66.6**	**53.5**	-	49M

transformations on the weights can be cached once, and then we never need to perform them again. For this reason, most methods do not have any overhead at all over standard convolutions at inference time. In particular, a layer of methods AOL, BCOP, CPL, SLL or SOC does require memory of $M + P$, just like a standard convolution. (For a definition of M and P see the beginning of Appendix C.) The methods that apply the convolution in the Fourier domain do require more memory. In particular, applying a Cayley convolutions requires storing $M + \frac{1}{2}s^2c^2$ floats, and applying a LOT convolutions requires memory to store $M + s^2c^2$ floats. The memory requirements of doing a forward pass through the whole model should be given as the sum of memory required to store <u>all</u> (transformed) model parameters and the memory required to store the <u>largest</u> activation during the forward pass.

D. Comparison with SOTA

In this section, we report state-of-the-art results from the literature. In contrast to our comparison, the runs reported often use larger architectures and longer training times. Find results in Table Tab. 4.

E. Experimental Setup

In addition to theoretically analyzing different proposed layers, we also do an empirical comparison of those layers. In order to allow for a fair and meaningful comparison, we try to fix the architecture, loss function and optimizer, and evaluate all proposed layers with the same setting. From the data in Table 1 we know that different layers will have very different throughputs. In order to have a fair comparison despite of that, we limit the total training time instead of fixing a certain amount of training epochs. We report results of training for 2h, 10h as well as 24h. We describe the chosen setting below.

E.1. Architecture

We show the architecture used for our experiments in Tables 5 and 6. It is a standard convolutional architecture, that doubles the number of channels whenever the resolution is reduced. Note that we exclusively use convolutions with the same input and output size as an attempt to make the model less dependent on the initialization used by the convolutional layers. We use kernel size 3 in all our main experiments. The layer *Zero Channel Padding* in Table 5 just appends channels with value 0 to the input, and the layer *First Channels(c)* outputs only the first c channels, and ignores the rest. Finally, the layer *Pixel Unshuffle* (implemented in *PyTorch*) takes each $2 \times 2 \times c$ patches of an image and reshapes them into size $1 \times 1 \times 4c$.

For each 1-Lipschitz layer, we also test architectures of different sizes. In particular, we define 4 categories of models based on the number of parameters. We call those categories XS, S, M and L. See Table 7 for the exact numbers. In this table we also report the *width parameter* w that ensures our architecture has the correct number of parameters.

Remark 2. For most methods, the number of parameters per layer are about the same. There are two exceptions, BCOP and Sandwich. BCOP parameterizes the convolution kernel with c input channels and c output channels using a matrix of size

Table 5. **Architecture.** It depends on width parameter w, kernel size k ($k \in \{1, 3\}$) and the number of classes c. For details of the *Downsize Block* see Tab. 6.

Layer name	Output size
Input	$32 \times 32 \times 3$
Zero Channel Padding	$32 \times 32 \times w$
Conv (1×1 kernel size)	$32 \times 32 \times w$
Activation	$32 \times 32 \times w$
Downsize Block(k)	$16 \times 16 \times 2w$
Downsize Block(k)	$8 \times 8 \times 4w$
Downsize Block(k)	$4 \times 4 \times 8w$
Downsize Block(k)	$2 \times 2 \times 16w$
Downsize Block(1)	$1 \times 1 \times 32w$
Flatten	$32w$
Linear	$32w$
First Channels(c)	c

Table 6. Downsize Block(k) with input size $s \times s \times t$:

	Layer name	Kernel size	Output size
$5 \times \{$	Conv	$k \times k$	$s \times s \times t$
	Activation	-	$s \times s \times t$
	First Channels	-	$s \times s \times t/2$
	Pixel Unshuffle	-	$s/2 \times s/2 \times 2t$

Table 7. Number of parameters for different model sizes, as well as the *width parameter* w such that the architecture in Tab. 5 has the correct size.

Size	Parameters (millions)	w
XS	$1 < p < 2$	16
S	$4 < p < 8$	32
M	$16 < p < 32$	64
L	$64 < p < 128$	128

$c \times c$ and $2(k-1)$ matrices of size $c \times c/2$. Therefore, the number of parameters of a convolution using BCOP is kc^2, less than the k^2c^2 parameters of a plain convolution. The Sandwich layer has about twice as many parameters as the other layers for the same width, as it parameterizes two weight matrices, A and B in Equation (12), per layer.

E.2. Optimizer

We use SGD with a momentum of 0.9 for all experiments. We also used a learning rate schedule. We choose to use *OneCycleLR*, as described by [38], with default values as in *PyTorch*. We set the batch size to 256 for all the datasets except for Imagenette where we choose 64 for memory-capacity reasons.

E.3. Training Time

On of our main goals is to evaluate what is the best model to use given a certain time budget. In order to do this, we measure the time per epoch as described in Section 4.1 on an A100 GPU with 80GB memory for different methods and different model sizes. Then we estimate the number of epochs we can do in our chosen time budget of either 2h, 10h or 24h, and use that many epochs to train our models. The amount of epochs corresponding to the given time budget is summarized in Table 8.

Table 8. Budget of training epochs for different model sizes, layer types and datasets. The batch size is 64 for Imagenette, and 256 for other datasets. The training time is set to 2h for all datasets and model sizes.

	CIFAR				TinyImageNet				Imagenette		
	XS	S	M	L	XS	S	M	L	S	M	L
AOL	837	763	367	83	223	213	123	34	136	108	54
BCOP	127	125	94	24	50	50	39	11	63	51	16
CPL	836	797	522	194	240	194	148	63	136	108	62
Cayley	356	214	70	17	138	86	30	8	81	47	-
LOT	222	68	11	-	83	29	5	-	33	-	-
SLL	735	703	353	79	242	194	118	32	134	110	59
SOC	371	336	201	77	122	87	63	27	98	68	28
Param.s (M)†	1.57	6.28	25.12	100.46	1.58	6.29	25.16	100.63	6.30	25.18	100.68

† BCOP has less parameters overall, see Remark 2.

E.4. Hyperparameter Random Search

The learning rate and weight decay for each setting (model size, method and dataset) was tuned on the validation set. For each method we did hyperparameter search by training for $2h$ (corresponding number of epochs in Table 8). We did 16 runs with learning-rate of the form 10^x, where x is sampled uniformly in the interval $[-4, -1]$, and with weight-decay of the form 10^x, where x is sampled uniformly in the interval $[-5.5, -3.5]$. Finally, we selected the learning rate and weight decay corresponding to the run with the highest validation certified robust accuracy for radius $36/255$. We use these hyperparameters found also for the experiments with longer training time.

E.5. Datasets

We evaluate on four different datasets, CIFAR-10, CIFAR-100 [21], Tiny ImageNet [23] and Imagenette [16]. For CIFAR-10 and CIFAR-100 we use the architecture described in Table 5. Since the architectures are identical, so are time- and memory requirements, and therefore also the epoch budget. As preprocessing we subtract the dataset channel means from each image. As data augmentation at training time we apply random crops (4 pixels) and random flipping.

In order to assess the behavior on larger images, we replicate the evaluation on the Tiny ImageNet dataset [23]: a subset of 200 classes of the ImageNet [12] dataset, with images scaled to have size 64×64. In order to allow for the larger input size of this dataset, we add one additional *Downsize Block* to our model. We also divide the width parameter (given in Table 7) by 2 to keep the amount of parameters similar. We again subtract the channel mean for each image. As data augmentation we we us *RandAugment* [11] with 2 transformations of magnitude 9 (out of 31).

As a higher resolution dataset we use Imagenette. It is a subset of 10 classes from Imagenet, chosen to be easily distinguishable. We use the version with images with their shorter side resized to 320 pixels. At training time we first apply *RandAugment* (magnitude 9) to those images, than take random crops of size 256×256, and finally subtract the dataset channel means. At inference time we use center crops of the same size. We also adjust our model slightly and use 8 *Downsize Blocks* in total, with initial width set to 4, 8 or 16 (for model size S, M and L), in order conform to the size guide given in Table 7. Finally, we reduced the batch size to 64 in order reduce the memory requirements. Note that for the large input size of this dataset, perturbations with l_2 norm bounded by $36/255$ are tiny (e.g. equivalent to perturbing about 1% of the pixels by a value of 1). However, for consistency with loss function etc. we choose to also use this radius for Imagenette. Furthermore, a large proportion of the time when training on Imagenette is used for loading the images on the GPU. We purposefully included this time in our in the determination of the epoch budget, since data loading is of course required for training a model. In order to determine the epoch budget on Imagenette, we used an NVIDIA A40 GPU with 48 GB memory, and we did 12 runs for the hyperparameter search.

E.6. Metrics

As described in Section 4.1 we evaluate the methods in terms of three main metrics. The throughput, the memory requirements as well and the certified robust accuracy a model can achieve in a fixed amount of time.

The evaluation of the *throughput* is performed on an `NVIDIA A100 80GB PCIe GPU`[*]. We measure it by averaging the time it takes to process 100 batches (including forward pass, backward pass and parameter update), and use this value to calculate the average number of examples a model can process per second. In order to estimate the inference throughput, we first evaluate and cache all calculations that do not depend on the input (such as power iterations on the weights). With this we measure the average time of the forward pass of 100 batches, and calculate the throughput from that value.

F. Batch Activation Variance

As one (simple to compute) sanity check that the models we train are actually 1-Lipschitz, we consider the *batch activation variance*. For layers that are 1-Lipschitz, we show below that the batch activation variance cannot increase from one layer to the next. This gives us a mechanism to detect (some) issues with trained models, including numerical ones, conceptual ones as well as problems in the implementation.

To compute the batch activation variance we consider a mini-batch of inputs, and for this mini-batch we consider the outputs of each layer. Denote the outputs of layer l as $a_1^{(l)}, \dots, a_b^{(l)}$, where b is the batch size. Then we set

$$\mu^{(l)} = \frac{1}{b}\sum_{i=1}^{b} a_i^{(l)} \tag{14}$$

$$\mathrm{BatchVar}_l = \frac{1}{b}\sum_{i=1}^{b} \|a_i^{(l)} - \mu^{(l)}\|_2^2, \tag{15}$$

where the l_2 norm is calculated based on the flattened tensor. Denote layer l as f_l. Then we have that

$$\begin{aligned}
\mathrm{BatchVar}_{l+1} &= \frac{1}{b}\sum_{i=1}^{b} \|a_i^{(l+1)} - \mu^{(l+1)}\|_2^2 && (16)\\
&\leq \frac{1}{b}\sum_{i=1}^{b} \|f_l(a_i^{(l)}) - f_l(\mu^{(l)})\|_2^2 && (17)\\
&\leq \frac{1}{b}\sum_{i=1}^{b} \|a_i^{(l)} - \mu^{(l)}\|_2^2 && (18)\\
&= \mathrm{BatchVar}_l\,. && (19)
\end{aligned}$$

Here, for the first inequality we use that (by definition) $a_i^{(l+1)} = f_l(a_i^{(l)})$ and that the term $\sum_{i=1}^{n} \|a_i^{(l+1)} - x\|_2^2$ is minimal for $x = \mu^{(l+1)}$. The second inequality follows from the 1-Lipschitz property. The equation above shows that the batch activation variance can not increase from one layer to the next for 1-Lipschitz layers. Therefore, if we see an increase in experiments that shows that the layer is not actually 1-Lipschitz.

As a further check that the layers are 1-Lipschitz we also apply (convolutional) power iteration to each linear layer after training.

G. Further Experimental Results

In this section, further experiments –not presented in the main paper— can be found.

G.1. Certified robust accuracy

A plot of the certified robust accuracy of different models sorted by value can be found in Figure 6.

[*]https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/PB-10577-001_v02.pdf

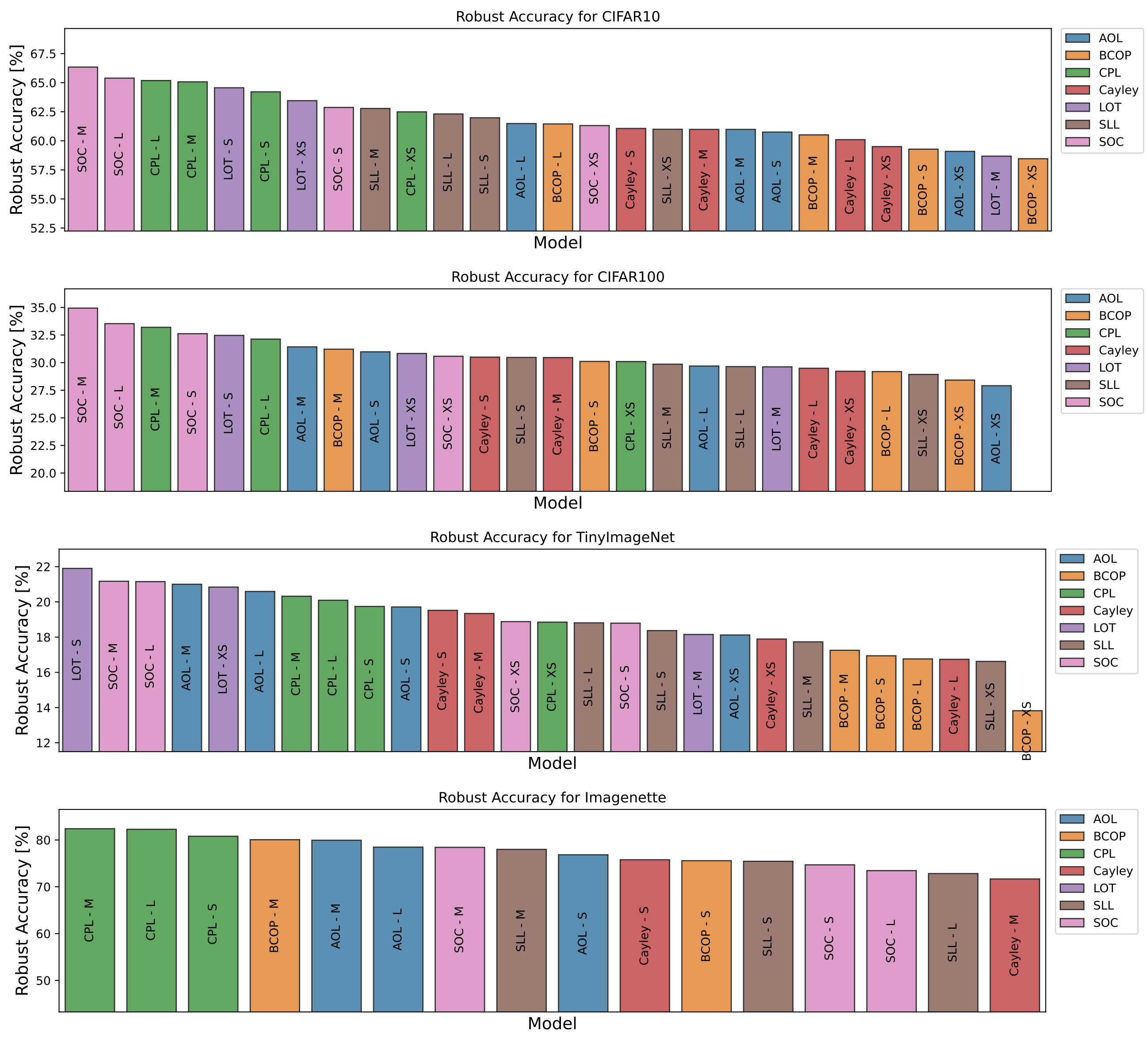

Figure 6. **Certified robust accuracy** ($\epsilon = 36/255$) in decreasing order. Note that the axes do not start at 0.

G.2. Different training time budgets

In this section we report the experimental results for three different training budgets: $2h$, $10h$ and $24h$. See the results in Table 9 (CIFAR-10), Table 10 (CIFAR-100), Table 11 (Tiny ImageNet) and Table 12 (Imagenette). Each of those tables also reports the best learning rate and weight decay found by the random search for each setting. Furthermore, a different representation of the impact of the training time on the robust accuracy can be found in Figure 7.

G.3. Time and Memory Requirements on Tiny ImageNet

See Appendix G.3 for an evaluation of time and memory usage for Tiny ImageNet dataset. The models used on CIFAR-10 and the ones on Tiny ImageNet are identical up to one convolutional block, therefore also the results in Appendix G.3 are similar to the results on CIFAR-10 reported in the main paper.

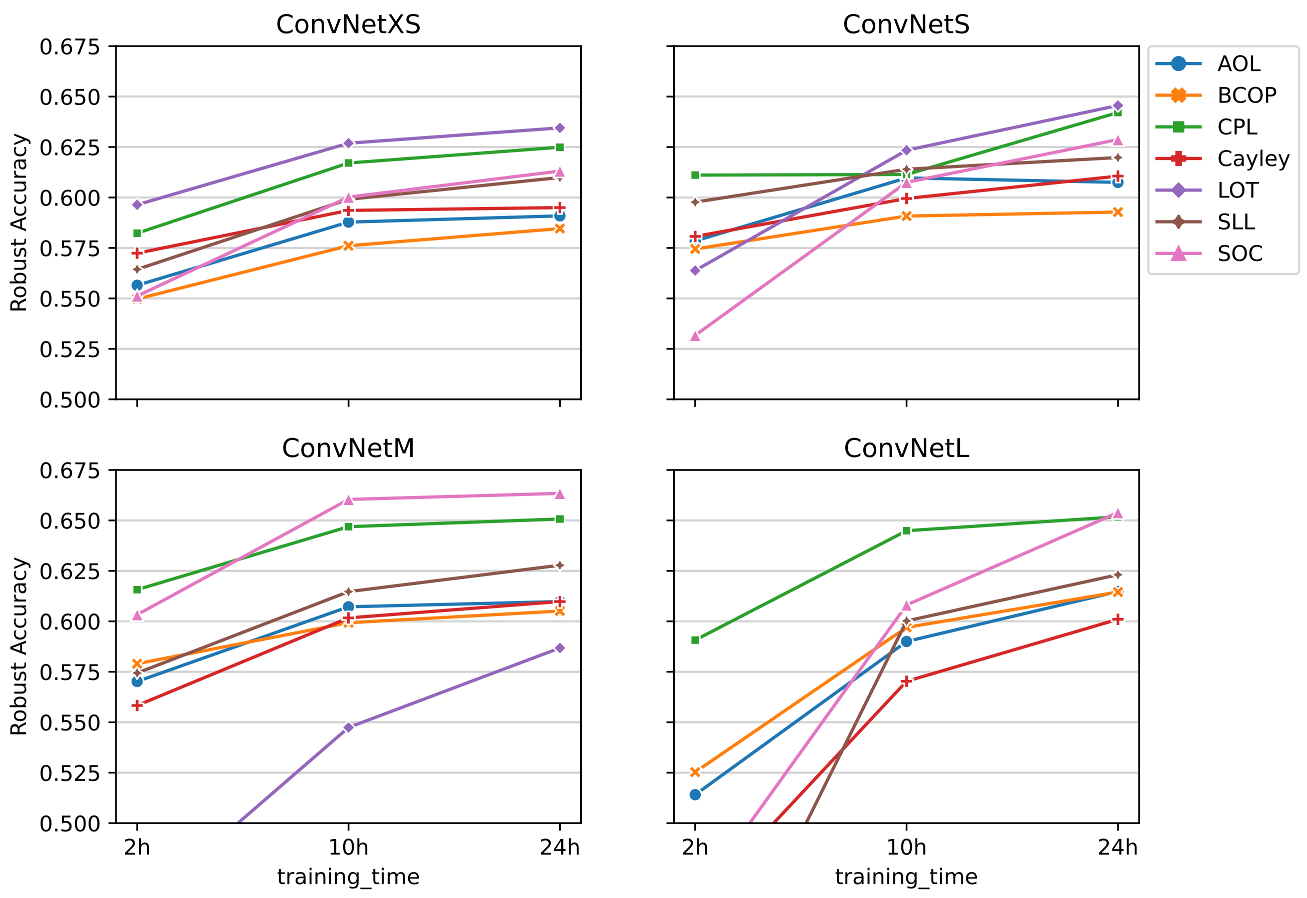

Figure 7. Line plots of the robust accuracy for various methods where models are training with different time budgets

G.4. Kernel size 1×1

For CIFAR-10 dataset, we tested models where convolutional layers have a kernel size of 1×1, to evaluate if the quicker epoch time can compensate for the lower number of parameters. In almost all cases the answer was negative, the version with 3×3 kernel outperformed the one with the 1×1 kernel.We therefore do not recommend reducing the kernel size. See Table 13 for a detailed view of the accuracy and robust accuracy.

G.5. Clean Accuracy

As a reference, we plotted the accuracy of different models in Figure 9. The rankings by accuracy are similar to the rankings by certified robust accuracy. One difference is that the Cayley method performs better relative to other methods when measured in terms of accuracy.

H. Additional Discussion

We will provide additional discussions on the results that we have described in the main paper, and some speculation what contributes to a well performing methods: (See also Figure 10.)

H.1. Detailed Discussion of the results

All layer types considered in this paper have comparable expressive power. Therefore, all networks could reach comparable accuracy if trained to convergence. However, training Lipschitz networks to convergence is typically not possible, because the training loss decreases very slowly (logarithmic in the number of epochs). This is the reason why we adopt the setting of time budgets.

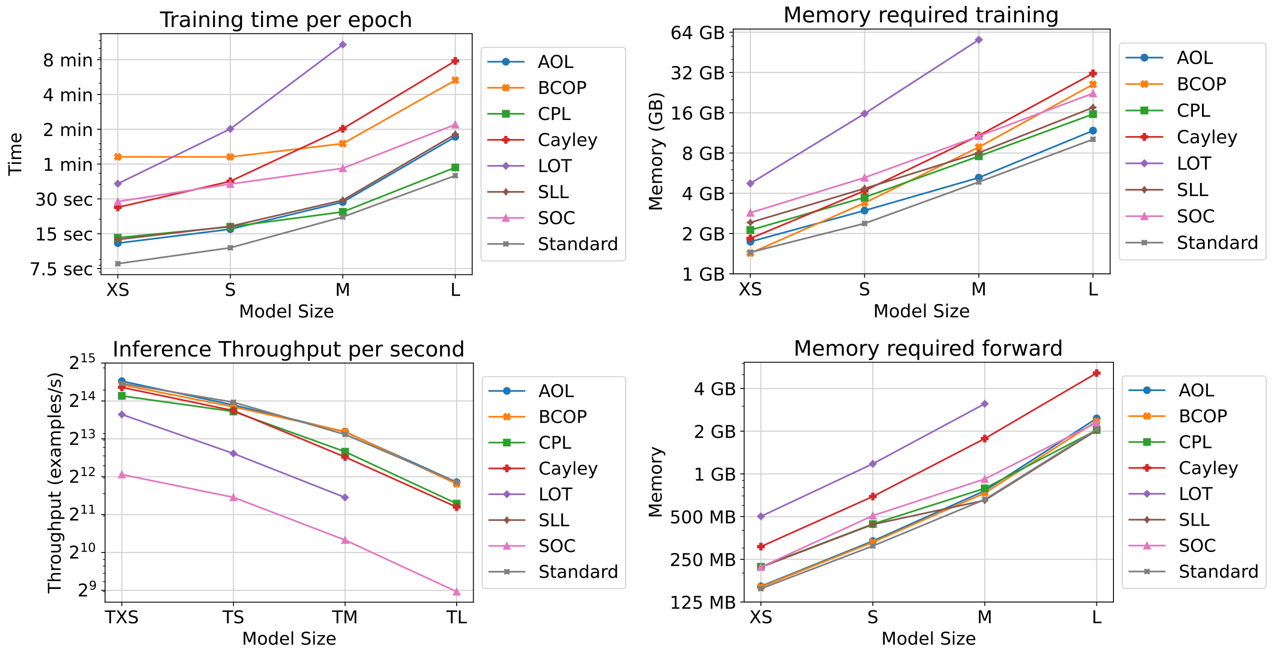

Figure 8. Measured time and memory requirements on Tiny ImageNet.

Different 1-Lipschitz layers differ in their parametrizations, which influences the loss landscape and gradient properties. Our results suggest that the main factor determining a method's robust accuracy is how easily it can represent the identity map. Layers that can proxy as skip connections (CPL, SLL, SOC) generally perform better than layers that do not have this ability, but use close-to-identity initialization (AOL, LOT), which themselves perform better than layers that do neither (BCOP, Cayley). Presumably this is because MaxMin activations reduce the batch activation variance in the forward pass when alternated with non-identity layers. Low variance implies small difference in score values, and this in turn causes low certified robust accuracy.

Note that the three methods CPL, SLL and SOC divide by an upper bound of the Lipschitz constant in their calculation at some point, and all three methods have the property that if the upper bound is loose they do approximate the identity map. So by setting the parameters to any value that makes the bound loose, the network can approximate the identity map. Since the upper bounds are usually only tight for a small subspace of parameter matrices, we expect that a large learning rate does not hurt the performance too much, since the individual layers will just end up approximating an identity map. This seems like a very useful property for a network to possess, and we expect that this is one important reason behind the good performance of those methods.

Our results also allow ruling out some other possible explanation: one might suspect that AOL, CPL, and SLL suffer from vanishing gradients, because they do not enforce exact orthogonality but only enforce that the Lipschitz constant is bounded by 1. Our experiments, however, do not show evidence of this. Also, one might suspect that slower methods perform worse, because they allow fewer epochs for a given time budget. Again, our experiments do not support this. Two relative slow methods (SOC, LOT) are among the best ones.

H.2. Influence of Image Resolution:

The results on Imagenette (Table 12) highlight some different properties of the methods. First note that when training for 2 hours the performance of different models varies drastically, but when trained for longer (e.g. 24 hours) most methods reach more similar performance. Among the methods, CPL clearly performs best. AOL also perform well, SLL is competitive in particular for shorter training times, and BCOP does well at least when trained for 24 hours. Note that the calculations performed by those methods are independent of the image resolution; the methods rescale or construct the kernel directly. So in relation to applying the convolution, the rescaling becomes much cheaper to calculate. This is not the case for SOC, and whilst it was a top performer on the datasets with low image resolution, it is much worse compared to other methods on Imagenette.

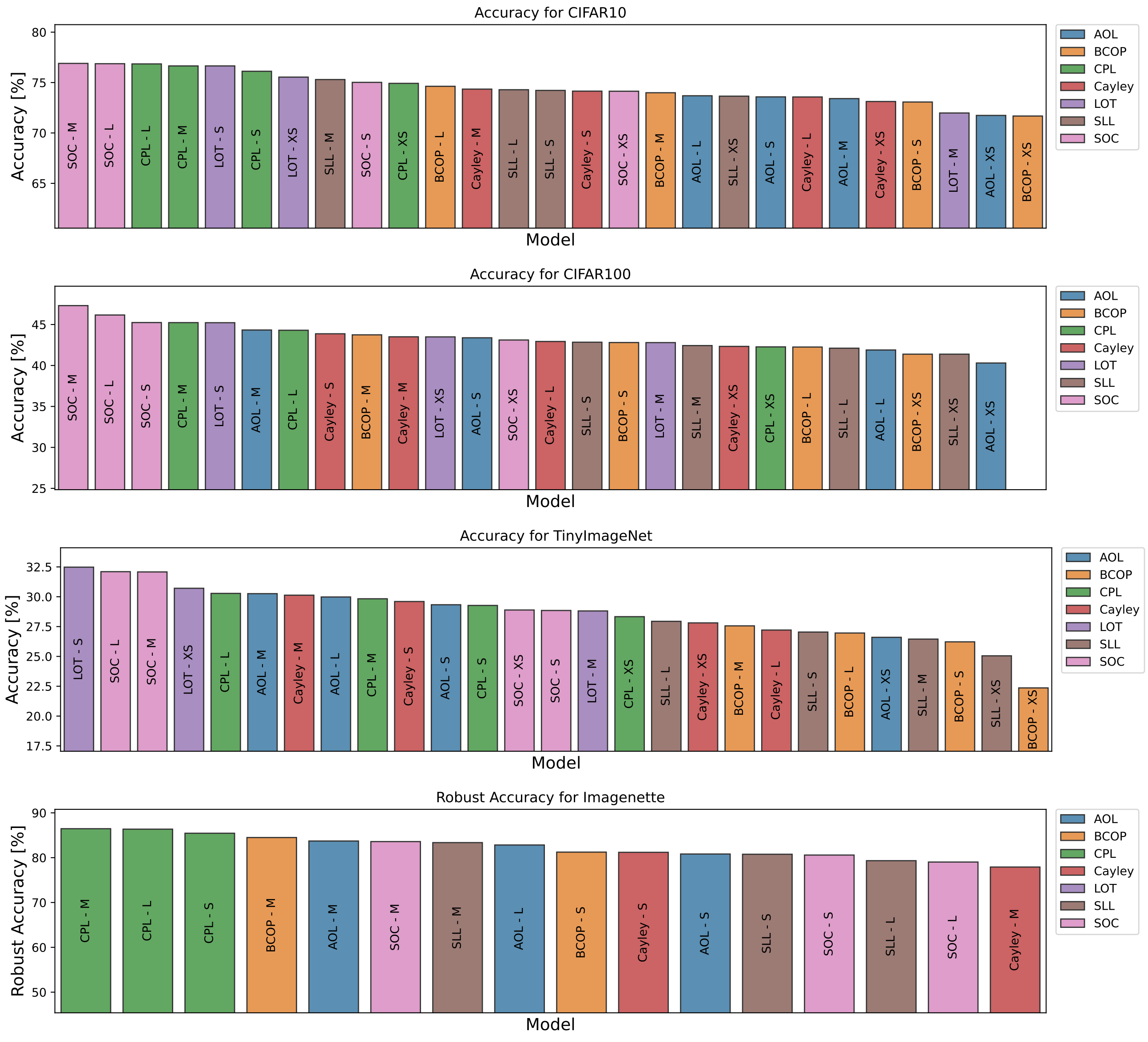

Figure 9. Barplots of clean accuracy of the models with the different layers, sorted by decreasing clean accuracy.

H.3. Potential future directions

Based on some of the observations above, there are plenty of interesting future directions. For example, we did not consider the influence of

- **Batch size:** The parameter transformation (by definition) only have to be done once per batch. So, in particular for computationally expensive methods, we expect that increasing the batch size could increase performance. For this then the required memory becomes an important factor.
- **Initialization:** As mentioned above, we do believe that initialization is a very important factor for 1-Lipschitz networks. We are very curious to see the influence being explored in future work.
- **Model depth:** In this work, we have only varied the model with, and left the depth constant. Maybe model depth is an important factor of how well some of the methods perform.
- **Optimizer:** Some methods might benefit from using a different optimizer, for example ADAM.
- **Data augmentation:** We used only simple standard data augmentation on CIFAR. Maybe more complex data augmenta-

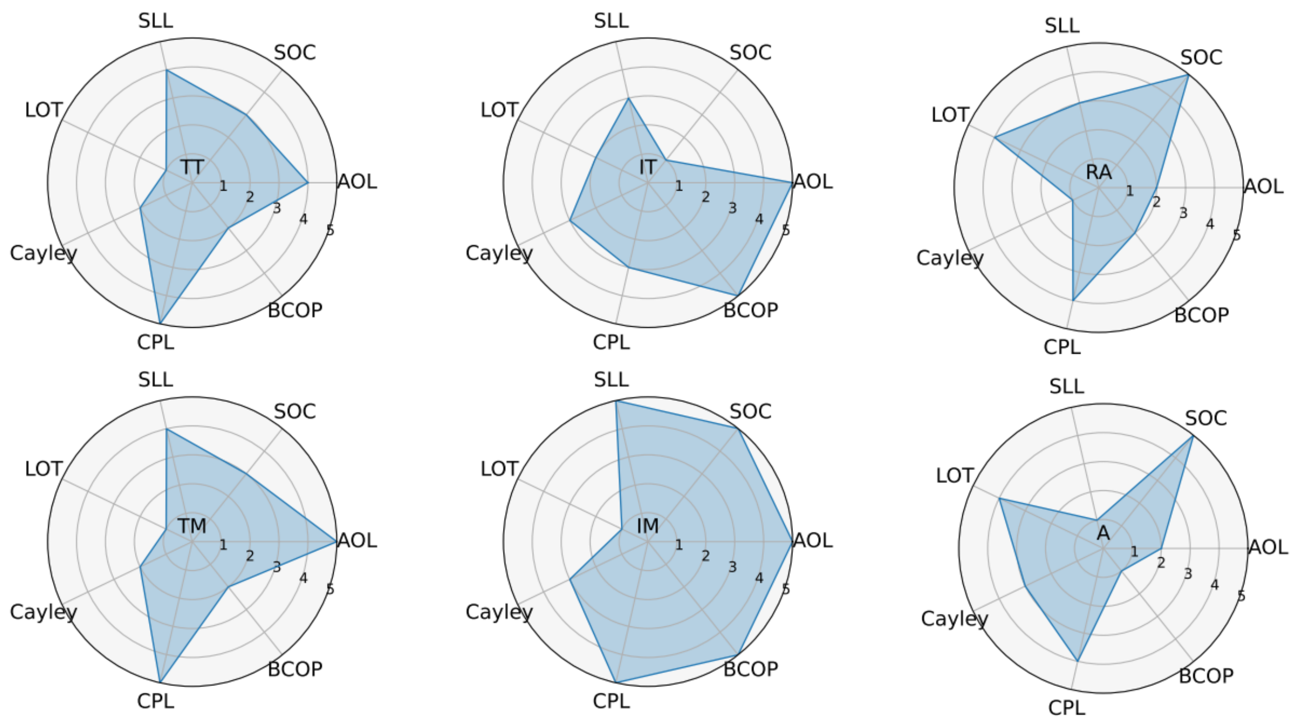

Figure 10. Overall performance of different methods summarized in a single number, for metrics training time per epoch (TT), inference time per epoch (IT), certified robust accuracy (RA), training memory requirements (TM), inference memory requirements (IM) as well as accuracy (A). Higher is better.

tion, potentially even using a generative model, as done by [1, 17], can change the relative performance of the methods compared.

Finally, different methods seem to have different strengths, it might be useful to look into combining them, for example by using different methods for early (high resolution) layers and later layers (with more channels).

I. Issues and observations

As already thoroughly analyzed in the dedicated section, some of the known methods in the literature have been omitted in the main paper since we faced serious concerns. Nevertheless, we also encountered some difficulties during the implementation of the methods that we did report in the main paper. The aim of this section is to highlight these difficulties, we hope this can open a constructive debate.

1. In the SLL [3] code, taken from the authors' repository, there is no attention to numerical errors that can easily happen from close-to-zero divisions during the parameter transformation. We solve the issue, by adopting the commonly used strategy, i.e. we included a factor of $1 \cdot 10^{-6}$ while dividing for the AOL-rescaling of the weight matrix. Furthermore, the code provided in the SLL repository only works for a kernel size of 3, we fixed the issue in our implementation.
2. The CPL method [31] features a high sensitivity to the initialization of the weights. Long training, e.g. 24-hour training, can sometimes result in *NaN* during the update of the weights. In that case we re-ran the models with different seeds.
3. Furthermore, there was no initialization method stated in the CPL paper, and also no code was provided. Therefore, we used the initialization from the similar SLL method.
4. During the training of Sandwich we faced some numerical errors. To investigate such errors, we tested a lighter version of the method — without the learnable rescaling Ψ — for the reason described in Remark 3, which shows that the rescaling Ψ inside the layer can be embedded into the bias term and hence the product $\Psi\Psi^{-1}$ can be omitted.
5. Similarly, for SLL, the matrix Q in Equation (8) does not add additional degrees of freedom to the model. Instead of having parameters W, Q and b we could define and optimize over $P = WQ^{-1}$ and $\tilde{b} = Q^{-1}b$. However, for our experiments we used the original parameterization.

6. The method Cayley [40], in the form proposed in the original paper, does not cache the — costly — transformation of the weight matrix whenever the layer is in inference mode. We fix this issue in our implementation.
7. Also, for method Cayley the algorithm stated in their paper differs from the version in their code. We used the version from the supplied code for our analysis as well as experiments.
8. The LOT method, [46], leverages 10 inner iterations for both training and inference in order to estimate the inverse of the square root with their proposed Newton-like method. Since the gradient is tracked for the whole procedure, the amount of memory required during the training is prohibitive for large models. Furthermore, since the memory is required for the parameter transformation, reducing the batch size does not solve this problem. In order to make the Large model (L) fit in the memory, we tested the LOT method with only 2 inner iterations. However, the performance in terms of accuracy and robust accuracy is not comparable to other strategies, hence we omitted it from our tables.

Remark 3. The learnable parameter Ψ of the sandwich layer corresponds to a scaling of the bias. In details, for each parameters A, B, b and $\Psi = \operatorname{diag}\left(e^{d_i}\right)$ there exists a rescaling of the bias $\tilde{b}$ such that

$$l(x) = \sqrt{2}A^T \Psi \mathrm{ReLU}\left(\sqrt{2}\Psi^{-1}Bx + b\right) = \sqrt{2}A^T \mathrm{ReLU}\left(\sqrt{2}Bx + \tilde{b}\right) \tag{20}$$

Proof. Observing that for each $\alpha > 0$ and $x \in \mathbb{R}$, $\mathrm{ReLU}\left(\alpha x\right) = \alpha \mathrm{ReLU}\left(x\right)$, and that

$$\forall \mathbf{x} \in \mathbb{R}^n, \quad \Psi^{-1}\mathbf{x} = \begin{bmatrix} e^{-d_1}x_1 \\ \vdots \\ e^{-d_n}x_n \end{bmatrix},$$

the following identity holds

$$\begin{aligned} l(x) &= \sqrt{2}A^T \Psi \mathrm{ReLU}\left(\sqrt{2}\Psi^{-1}Bx + b\right) \\ &= \sqrt{2}A^T \Psi \mathrm{ReLU}\left(\sqrt{2}\Psi^{-1}\left(\sqrt{2}Bx + \Psi b\right)\right) \\ &= \sqrt{2}A^T \Psi \Psi^{-1} \mathrm{ReLU}\left(\sqrt{2}Bx + \Psi b\right) \\ &= \sqrt{2}A^T \mathrm{ReLU}\left(\sqrt{2}Bx + \Psi b\right). \end{aligned} \tag{21}$$

Considering $\tilde{b} = \Psi b$ concludes the proof. □

J. Code

We often build on code provided with the original papers. This includes
- https://github.com/berndprach/AOL (AOL)
- https://github.com/ColinQiyangLi/LConvNet (BCOP)
- https://github.com/MILES-PSL/Convex-Potential-Layer (CPL)
- https://github.com/locuslab/orthogonal-convolutions (Cayley)
- https://github.com/AI-secure/Layerwise-Orthogonal-Training (LOT)
- https://github.com/acfr/LBDN (Sandwich)
- https://github.com/araujoalexandre/Lipschitz-SLL-Networks (SLL)
- https://github.com/singlasahil14/SOC (SOC)

We are grateful for authors providing code to the research community.

Table 9. Experimental results on **CIFAR-10**. We report the best learning rate and weight decay found by a random search, as well as accuracy and certified robust accuracy for radius $\epsilon = 36/355$, both for different training time budgets.

				Accuracy [%]			Robust Accuracy [%]		
Layer	Model	LR	WD	2h	10h	24h	2h	10h	24h
AOL	**XS**	$\mathbf{6 \cdot 10^{-2}}$	$\mathbf{4 \cdot 10^{-5}}$	68.5	71.7	71.7	55.6	58.8	**59.1**
	S	$\mathbf{1 \cdot 10^{-2}}$	$\mathbf{5 \cdot 10^{-5}}$	70.9	73.0	73.6	57.9	**61.0**	60.8
	M	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	70.3	73.4	73.4	57.0	60.7	**61.0**
	L	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{7 \cdot 10^{-5}}$	64.7	71.8	73.7	51.4	59.0	**61.5**
BCOP	**XS**	$\mathbf{7 \cdot 10^{-3}}$	$\mathbf{3 \cdot 10^{-4}}$	69.0	70.8	71.7	55.0	57.6	**58.5**
	S	$\mathbf{4 \cdot 10^{-3}}$	$\mathbf{8 \cdot 10^{-5}}$	70.6	72.5	73.1	57.5	59.1	**59.3**
	M	$\mathbf{6 \cdot 10^{-3}}$	$\mathbf{7 \cdot 10^{-6}}$	71.8	73.6	74.0	57.9	59.9	**60.5**
	L	$\mathbf{1 \cdot 10^{-3}}$	$\mathbf{9 \cdot 10^{-6}}$	67.6	73.2	74.6	52.5	59.7	**61.5**
CPL	**XS**	$\mathbf{3 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	71.7	74.3	74.9	58.2	61.7	**62.5**
	S	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	73.9	74.1	76.1	61.1	61.1	**64.2**
	M	$\mathbf{8 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	74.3	76.4	76.6	61.6	64.7	**65.1**
	L	$\mathbf{5 \cdot 10^{-2}}$	$\mathbf{3 \cdot 10^{-4}}$	72.6	76.5	76.8	59.1	64.5	**65.2**
Cayley	**XS**	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	71.3	73.2	73.1	57.2	59.4	**59.5**
	S	$\mathbf{1 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-5}}$	71.9	73.6	74.2	58.1	60.0	**61.1**
	M	$\mathbf{1 \cdot 10^{-2}}$	$\mathbf{6 \cdot 10^{-6}}$	70.2	73.5	74.4	55.8	60.2	**61.0**
	L	$\mathbf{8 \cdot 10^{-3}}$	$\mathbf{2 \cdot 10^{-4}}$	61.3	71.1	73.6	45.9	57.0	**60.1**
LOT	**XS**	$\mathbf{8 \cdot 10^{-2}}$	$\mathbf{4 \cdot 10^{-6}}$	73.5	75.2	75.5	59.6	62.7	**63.4**
	S	$\mathbf{3 \cdot 10^{-2}}$	$\mathbf{7 \cdot 10^{-5}}$	70.5	75.0	76.6	56.4	62.3	**64.6**
	M	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{9 \cdot 10^{-6}}$	61.4	69.3	72.0	45.7	54.7	**58.7**
SLL	**XS**	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	69.9	73.1	73.7	56.4	59.9	**61.0**
	S	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	72.9	74.0	74.2	59.8	61.4	**62.0**
	M	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{9 \cdot 10^{-5}}$	70.5	74.4	75.3	57.4	61.5	**62.8**
	L	$\mathbf{6 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	56.7	72.7	74.3	38.9	60.0	**62.3**
SOC	**XS**	$\mathbf{5 \cdot 10^{-2}}$	$\mathbf{7 \cdot 10^{-6}}$	68.9	72.9	74.1	55.1	60.0	**61.3**
	S	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	67.3	73.3	75.0	53.2	60.8	**62.9**
	M	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	73.1	77.0	76.9	60.3	66.0	**66.3**
	L	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	62.3	73.8	76.9	46.2	60.8	**65.4**

Table 10. Experimental results on **CIFAR-100**. We report the best learning rate and weight decay found by a random search, as well as accuracy and certified robust accuracy for radius $\epsilon = 36/355$, both for different training time budgets.

				Accuracy [%]			Robust Accuracy [%]		
Layer	Model	LR	WD	2h	10h	24h	2h	10h	24h
AOL	**XS**	$\mathbf{3 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	37.9	40.1	40.3	26.6	**28.0**	27.9
	S	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	40.5	43.4	43.4	29.0	30.8	**31.0**
	M	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{6 \cdot 10^{-6}}$	40.5	43.5	44.3	28.4	31.1	**31.4**
	L	$\mathbf{6 \cdot 10^{-2}}$	$\mathbf{4 \cdot 10^{-5}}$	34.5	41.1	41.9	23.1	29.2	**29.7**
BCOP	**XS**	$\mathbf{8 \cdot 10^{-3}}$	$\mathbf{3 \cdot 10^{-4}}$	35.4	40.0	41.4	22.9	27.8	**28.4**
	S	$\mathbf{6 \cdot 10^{-3}}$	$\mathbf{3 \cdot 10^{-4}}$	37.8	42.1	42.8	24.5	29.5	**30.1**
	M	$\mathbf{2 \cdot 10^{-3}}$	$\mathbf{2 \cdot 10^{-4}}$	37.6	43.8	43.7	24.6	30.4	**31.2**
	L	$\mathbf{4 \cdot 10^{-3}}$	$\mathbf{1 \cdot 10^{-5}}$	29.2	40.3	42.2	17.3	27.2	**29.2**
CPL	**XS**	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{3 \cdot 10^{-5}}$	39.7	42.0	42.3	27.9	29.8	**30.1**
	S	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	42.1	1.0	1.0	**29.8**	0.0	0.0
	M	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{4 \cdot 10^{-5}}$	40.5	44.5	45.2	27.4	32.4	**33.2**
	L	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	40.1	43.9	44.3	27.3	31.5	**32.1**
Cayley	**XS**	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	41.1	41.6	42.3	27.9	29.2	**29.2**
	S	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	42.3	43.1	43.9	28.8	30.4	**30.5**
	M	$\mathbf{6 \cdot 10^{-3}}$	$\mathbf{4 \cdot 10^{-6}}$	38.6	43.3	43.5	25.3	29.9	**30.5**
	L	$\mathbf{5 \cdot 10^{-3}}$	$\mathbf{2 \cdot 10^{-5}}$	26.3	40.3	42.9	14.3	27.0	**29.5**
LOT	**XS**	$\mathbf{6 \cdot 10^{-2}}$	$\mathbf{3 \cdot 10^{-4}}$	42.7	43.8	43.5	29.4	**30.9**	30.8
	S	$\mathbf{5 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	40.3	45.2	45.2	27.2	31.8	**32.5**
	M	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{9 \cdot 10^{-5}}$	28.4	38.9	42.8	15.5	25.9	**29.6**
SLL	**XS**	$\mathbf{5 \cdot 10^{-2}}$	$\mathbf{6 \cdot 10^{-5}}$	37.9	40.9	41.4	25.9	**29.0**	28.9
	S	$\mathbf{1 \cdot 10^{-1}}$	$\mathbf{7 \cdot 10^{-5}}$	40.0	41.9	42.8	28.4	29.9	**30.5**
	M	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	39.3	41.9	42.4	27.0	**30.0**	29.9
	L	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{8 \cdot 10^{-5}}$	22.0	38.5	42.1	10.8	26.4	**29.6**
SOC	**XS**	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	40.7	43.1	43.1	27.7	29.7	**30.6**
	S	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	42.2	44.5	45.2	29.3	31.9	**32.6**
	M	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{3 \cdot 10^{-4}}$	41.8	46.2	47.3	28.8	33.5	**34.9**
	L	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{3 \cdot 10^{-5}}$	34.7	43.1	46.2	21.5	30.2	**33.5**

Table 11. Experimental results on **Tiny ImageNet**. We report the best learning rate and weight decay found by a random search, as well as accuracy and certified robust accuracy for radius $\epsilon = 36/355$, both for different training time budgets.

				Accuracy [%]			Robust Accuracy [%]		
Layer	Model	LR	WD	2h	10h	24h	2h	10h	24h
AOL	**XS**	$\mathbf{3 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	24.5	26.9	26.6	16.5	**18.4**	18.1
	S	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{5 \cdot 10^{-6}}$	24.9	27.3	29.3	16.8	18.5	**19.7**
	M	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{8 \cdot 10^{-6}}$	26.2	29.5	30.3	18.1	20.4	**21.0**
	L	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{8 \cdot 10^{-6}}$	22.1	27.7	30.0	12.8	18.8	**20.6**
BCOP	**XS**	$\mathbf{6 \cdot 10^{-4}}$	$\mathbf{5 \cdot 10^{-5}}$	11.7	20.4	22.4	4.7	11.6	**13.8**
	S	$\mathbf{7 \cdot 10^{-4}}$	$\mathbf{4 \cdot 10^{-5}}$	19.0	25.3	26.2	10.1	15.7	**16.9**
	M	$\mathbf{3 \cdot 10^{-4}}$	$\mathbf{1 \cdot 10^{-4}}$	20.0	25.4	27.6	10.5	15.8	**17.2**
	L	$\mathbf{1 \cdot 10^{-4}}$	$\mathbf{3 \cdot 10^{-4}}$	9.6	23.6	27.0	2.6	14.0	**16.8**
CPL	**XS**	$\mathbf{1 \cdot 10^{-1}}$	$\mathbf{9 \cdot 10^{-6}}$	26.5	27.5	28.3	16.3	17.5	**18.9**
	S	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{3 \cdot 10^{-5}}$	26.2	29.3	29.3	16.7	19.4	**19.7**
	M	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	26.3	29.4	29.8	16.7	19.5	**20.3**
	L	$\mathbf{6 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-5}}$	24.7	29.8	30.3	15.0	19.2	**20.1**
Cayley	**XS**	$\mathbf{8 \cdot 10^{-3}}$	$\mathbf{8 \cdot 10^{-5}}$	25.3	27.5	27.8	15.7	**17.9**	**17.9**
	S	$\mathbf{5 \cdot 10^{-3}}$	$\mathbf{7 \cdot 10^{-5}}$	25.6	29.3	29.6	15.8	19.1	**19.5**
	M	$\mathbf{4 \cdot 10^{-3}}$	$\mathbf{4 \cdot 10^{-5}}$	20.2	29.2	30.1	9.9	18.8	**19.3**
	L	$\mathbf{1 \cdot 10^{-3}}$	$\mathbf{3 \cdot 10^{-4}}$	5.7	22.1	27.2	0.9	11.9	**16.7**
LOT	**XS**	$\mathbf{3 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	28.1	31.1	30.7	18.2	20.4	**20.8**
	S	$\mathbf{5 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	27.1	32.0	32.5	16.3	21.6	**21.9**
	M	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{6 \cdot 10^{-6}}$	12.6	25.2	28.8	4.6	14.5	**18.1**
SLL	**XS**	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{3 \cdot 10^{-4}}$	24.2	25.3	25.1	14.9	**16.7**	16.6
	S	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	24.4	25.7	27.0	15.6	16.8	**18.4**
	M	$\mathbf{5 \cdot 10^{-2}}$	$\mathbf{5 \cdot 10^{-5}}$	17.0	26.2	26.5	7.8	17.0	**17.7**
	L	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	11.4	26.2	27.9	2.9	16.7	**18.8**
SOC	**XS**	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	25.5	28.7	28.9	15.7	18.7	**18.9**
	S	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	23.9	28.4	28.8	14.0	18.5	**18.8**
	M	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	25.7	31.1	32.1	15.5	20.4	**21.2**
	L	$\mathbf{6 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	20.3	30.1	32.1	10.1	19.7	**21.1**

Table 12. Experimental results on **Imagenette**. We report the best learning rate and weight decay found by a random search, as well as accuracy and certified robust accuracy for radius $\epsilon = 36/355$, both for different training time budgets.

				Accuracy [%]			Robust Accuracy [%]		
Layer	Model	LR	WD	2h	10h	24h	2h	10h	24h
AOL	S	$\mathbf{9 \cdot 10^{-03}}$	$\mathbf{1 \cdot 10^{-05}}$	71.0	78.9	80.8	62.5	74.3	76.8
	M	$\mathbf{5 \cdot 10^{-03}}$	$\mathbf{1 \cdot 10^{-04}}$	73.2	80.7	83.7	64.3	75.8	79.9
	L	$\mathbf{5 \cdot 10^{-03}}$	$\mathbf{1 \cdot 10^{-04}}$	69.7	79.2	82.8	59.7	74.2	78.5
BCOP	S	$\mathbf{9 \cdot 10^{-04}}$	$\mathbf{1 \cdot 10^{-04}}$	37.3	75.4	81.2	15.1	66.9	75.6
	M	$\mathbf{2 \cdot 10^{-03}}$	$\mathbf{2 \cdot 10^{-04}}$	36.4	77.8	84.5	15.6	71.3	80.1
	L	$\mathbf{7 \cdot 10^{-02}}$	$\mathbf{4 \cdot 10^{-06}}$	9.8	9.8	9.8	9.8	9.8	9.8
CPL	S	$\mathbf{9 \cdot 10^{-02}}$	$\mathbf{5 \cdot 10^{-05}}$	75.8	83.5	85.5	68.3	78.8	80.8
	M	$\mathbf{9 \cdot 10^{-02}}$	$\mathbf{3 \cdot 10^{-05}}$	79.5	84.8	86.5	73.5	80.3	82.4
	L	$\mathbf{7 \cdot 10^{-02}}$	$\mathbf{3 \cdot 10^{-05}}$	76.0	85.1	86.4	68.5	80.3	82.3
Cayley	S	$\mathbf{6 \cdot 10^{-04}}$	$\mathbf{3 \cdot 10^{-05}}$	60.9	78.0	81.2	45.4	71.5	75.8
	M	$\mathbf{1 \cdot 10^{-04}}$	$\mathbf{1 \cdot 10^{-05}}$	48.6	69.7	77.9	30.1	59.4	71.7
LOT	S	$\mathbf{7 \cdot 10^{-03}}$	$\mathbf{1 \cdot 10^{-04}}$	66.2	tbd	tbd	55.7	tbd	tbd
SLL	S	$\mathbf{7 \cdot 10^{-02}}$	$\mathbf{5 \cdot 10^{-06}}$	70.6	77.6	80.8	60.6	70.1	75.4
	M	$\mathbf{7 \cdot 10^{-02}}$	$\mathbf{6 \cdot 10^{-06}}$	71.7	80.4	83.4	62.0	74.3	78.0
	L	$\mathbf{5 \cdot 10^{-02}}$	$\mathbf{5 \cdot 10^{-06}}$	64.7	75.4	79.3	51.0	67.2	72.8
SOC	S	$\mathbf{9 \cdot 10^{-03}}$	$\mathbf{7 \cdot 10^{-05}}$	60.7	77.2	80.6	45.1	69.7	74.7
	M	$\mathbf{2 \cdot 10^{-02}}$	$\mathbf{9 \cdot 10^{-05}}$	63.4	79.5	83.6	50.4	72.2	78.4
	L	$\mathbf{8 \cdot 10^{-03}}$	$\mathbf{3 \cdot 10^{-04}}$	52.5	71.7	79.0	34.1	62.2	73.5

Table 13. **Kernel size** 1×1, on CIFAR-10, for different time budgets. We report accuracy and certified robust accuracy for radius $\epsilon = 36/355$, as well as the best learning rate and weight decay found by a random search.

				Accuracy [%]			Robust Accuracy [%]		
Layer	Model	LR	WD	2h	10h	24h	2h	10h	24h
AOL	**XS**	$\mathbf{8 \cdot 10^{-2}}$	$\mathbf{7 \cdot 10^{-6}}$	66.5	67.9	68.0	52.1	53.9	**54.3**
	S	$\mathbf{5 \cdot 10^{-2}}$	$\mathbf{4 \cdot 10^{-5}}$	67.7	69.0	69.7	53.7	55.1	**55.6**
	M	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	67.5	69.4	70.0	54.1	55.7	**56.7**
	L	$\mathbf{3 \cdot 10^{-2}}$	$\mathbf{8 \cdot 10^{-5}}$	64.3	68.7	69.6	50.0	54.8	**56.1**
BCOP	**XS**	$\mathbf{1 \cdot 10^{-2}}$	$\mathbf{7 \cdot 10^{-5}}$	65.3	67.5	68.6	51.3	53.4	**54.5**
	S	$\mathbf{4 \cdot 10^{-3}}$	$\mathbf{2 \cdot 10^{-5}}$	66.9	69.1	69.5	52.6	54.3	**55.4**
	M	$\mathbf{8 \cdot 10^{-3}}$	$\mathbf{5 \cdot 10^{-6}}$	67.3	69.8	70.3	52.8	55.5	**56.1**
	L	$\mathbf{4 \cdot 10^{-3}}$	$\mathbf{2 \cdot 10^{-4}}$	64.0	68.6	69.6	48.9	54.3	**55.8**
BnB	**XS**	$\mathbf{8 \cdot 10^{-3}}$	$\mathbf{1 \cdot 10^{-4}}$	66.1	66.3	66.4	**52.1**	51.2	51.5
	S	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	67.9	69.8	69.7	53.1	**55.8**	55.3
	M	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{4 \cdot 10^{-6}}$	67.5	70.3	71.0	52.8	56.0	**56.8**
	L	$\mathbf{3 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	66.0	66.2	61.0	51.5	**51.7**	45.5
CPL	**XS**	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	69.2	71.5	72.0	55.5	58.6	**59.1**
	S	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{9 \cdot 10^{-5}}$	70.6	71.2	10.0	57.8	**58.2**	0.0
	M	$\mathbf{1 \cdot 10^{-1}}$	$\mathbf{2 \cdot 10^{-4}}$	69.7	10.0	10.0	**56.1**	0.0	0.0
	L	$\mathbf{8 \cdot 10^{-2}}$	$\mathbf{9 \cdot 10^{-6}}$	67.8	72.3	73.8	54.0	59.4	**61.1**
Cayley	**XS**	$\mathbf{1 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	65.3	67.4	67.9	50.7	52.9	**53.8**
	S	$\mathbf{1 \cdot 10^{-2}}$	$\mathbf{4 \cdot 10^{-6}}$	65.9	68.4	68.8	51.3	53.8	**54.5**
	M	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	63.5	67.5	69.1	48.8	53.4	**55.0**
	L	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{8 \cdot 10^{-5}}$	57.4	64.7	67.2	41.4	49.9	**52.9**
LOT	**XS**	$\mathbf{2 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	65.8	68.0	69.3	51.1	53.9	**54.9**
	S	$\mathbf{5 \cdot 10^{-2}}$	$\mathbf{3 \cdot 10^{-4}}$	60.0	68.1	68.8	44.4	54.3	**54.5**
SLL	**XS**	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{1 \cdot 10^{-4}}$	67.9	70.2	10.0	54.8	**57.3**	0.0
	S	$\mathbf{3 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	69.1	70.4	10.0	55.9	**57.1**	0.0
	M	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	69.0	10.0	10.0	**55.5**	0.0	0.0
	L	$\mathbf{9 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-4}}$	62.9	69.5	69.8	48.7	55.6	**56.1**
SOC	**XS**	$\mathbf{7 \cdot 10^{-2}}$	$\mathbf{2 \cdot 10^{-5}}$	65.0	67.2	67.5	49.2	52.1	**52.3**
	S	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{6 \cdot 10^{-6}}$	65.4	68.1	68.7	49.9	52.5	**54.2**
	M	$\mathbf{3 \cdot 10^{-2}}$	$\mathbf{7 \cdot 10^{-5}}$	66.0	69.2	70.5	50.5	54.3	**55.6**
	L	$\mathbf{4 \cdot 10^{-2}}$	$\mathbf{9 \cdot 10^{-5}}$	64.0	69.2	70.4	48.5	54.3	**55.9**