

A Criticism of DensEMANN

This appendix includes some additional experiments aimed at inspecting DensEMANN with a more critical eye. Through them, we aim to evaluate the true usefulness of some of its inner mechanisms—as well as of the algorithm itself.

An initial criticism of DensEMANN can be done by comparing it to other growing-based NAS algorithms in the literature. From this comparison, we can point out the following drawbacks:

- Its workflow is not based on a formal heuristic such as evolutionary computation [19, 14] or steepest descent [29], although the macro-algorithm is reminiscent of hill climbing.
- It does not explicitly manage the network’s size and resource consumption, for instance, through user-set constraints [29] or multi-objective optimization [19].
- Unlike more open approaches like [13], its choice of growing/pruning operations is very narrow, and limited by the DenseNet architectural paradigm.
- It does not use network morphisms, either exact [25, 26] or approximate [19], and so the network is not guaranteed to preserve its behaviour after growing and pruning operations.

On the other hand, we can also point out the following assets for DensEMANN:

- Its seed architecture is truly *minimal*—a single dense block with a single layer—in contrast with e.g. [10] where simplified human-designed architectures are used as seeds. A minimal seed conditions the search to start with small architectures, and allows for quicker training and growing in early epochs.
- The network’s growth, and thus its final size and resource consumption, is implicitly limited by an “in-supervised” minimal growing approach and an accuracy-based stopping criterion.
- Its cascade connection scheme (inspired by DenseNet [17], EMANN [16] and CC-Alg [22]) foments the complementarity of incrementally added layers.
- A special *complementarity* weight initialization method enforces the co-adaptation of new and old convolution filters in the network’s last layer.

For the experiments reported in this appendix, we used the same two computation environments as in the main paper (an MSi GT76 laptop and an internal cluster) as well as an extra third environment, an MSi GT75 Titan 8RG laptop with the following specifications: Windows 10 Home (64-bit) OS, Intel Core i7-8850H CPU (2.60 GHz), NVIDIA GeForce GTX 1080 GPU, 32.0 GB RAM (31.9 GB usable). Python is v3.6.7, PyTorch is v1.10.1+cu113.

In the tables in this appendix, the GPU times reported in black were fully obtained on the MSi GT76, those reported in *italized purple* were fully obtained on the internal cluster, and those reported in *italized blue-gray* are highly unreliable average times made out of runs in different environments (this only happens in Section A.2, more details in that section).

The total computation time for all the experiments in this appendix was 29.16 GPU days (16.60 days on the MSi GT76, 11.52 days on the internal cluster, and 1.04 days on the MSi GT75). If we only take into account the experiments not already in the main paper, the computation time for the appendix was 22.67 GPU days (12.99 days on the MSi GT76, 8.64 days on the internal cluster, and 1.04 days on the MSi GT75).

Below are the computation times for each section of the appendix:

- A.1.1: 13.97 GPU days (8.67 on the MSi GT76, 5.30 on the cluster). For experiments not already in the main paper, 9.09 GPU days (5.86 on the MSi GT76, 3.23 on the cluster).
- A.1.2: 5.51 GPU days (2.62 on the MSi GT76, 2.89 on the cluster). For experiments not already in the main paper, 3.90 GPU days (1.81 on the MSi GT76, 2.09 on the cluster).
- A.2: 9.68 GPU days (5.32 on the MSi GT76, 3.32 on the cluster, and 1.04 on the MSi GT75).

Table 3: Ablation study on DensEMANN’s block replication and use of cutout: average performance for growing and training DenseNet-BC on benchmark datasets

Dataset	DensEMANN setting	GPU execution time (hours)	GPU inference time (seconds)	Num. layers per block	Trainable parameters (k)
CIFAR-10	w/o all	7.95 ± 1.19	2.86 ± 0.15	6.2 ± 0.8	55.61 ± 9.19
	w/ cutout	9.83 ± 0.43	2.81 ± 0.08	5.8 ± 0.4	52.61 ± 5.55
	w/ repl.	10.34 ± 1.27	3.15 ± 0.25	5.6 ± 1.1	177.79 ± 41.74
	w/ repl. + cutout	13.48 ± 2.72	3.21 ± 0.36	5.8 ± 1.6	186.36 ± 56.68
Fashion-MNIST	w/o all	2.09 ± 0.71	2.65 ± 0.49	1.4 ± 0.5	11.03 ± 2.98
	w/ cutout	3.85 ± 1.54	2.99 ± 0.29	2.2 ± 1.6	15.30 ± 9.07
	w/ repl.	4.23 ± 0.54	3.34 ± 0.57	1.4 ± 0.5	34.71 ± 15.00
	w/ repl. + cutout	6.55 ± 1.80	3.98 ± 0.35	2.2 ± 1.3	51.84 ± 25.51
SVHN	w/o all	1.62 ± 0.11	10.94 ± 0.35	9.8 ± 0.8	78.85 ± 9.94
	w/ cutout	1.75 ± 0.70	10.19 ± 0.88	9.0 ± 4.1	72.24 ± 34.20
	w/ repl.	1.97 ± 0.60	11.58 ± 1.10	7.6 ± 3.0	214.57 ± 111.29
	w/ repl. + cutout	3.39 ± 0.26	13.36 ± 0.33	11.0 ± 1.2	339.81 ± 63.39

Table 3: Ablation study on DensEMANN’s block replication and use of cutout: average performance for growing and training DenseNet-BC on benchmark datasets, *continued*

Dataset	DensEMANN setting	Validation set		Test set	
		Acc. (%)	Loss	Acc. (%)	Loss
CIFAR-10	w/o all	86.70 ± 1.35	0.40 ± 0.04	86.09 ± 1.28	0.43 ± 0.03
	w/ cutout	82.07 ± 0.81	0.53 ± 0.03	85.33 ± 0.71	0.46 ± 0.02
	w/ repl.	92.20 ± 0.61	0.23 ± 0.02	92.13 ± 0.56	0.25 ± 0.01
	w/ repl. + cutout	90.06 ± 1.38	0.30 ± 0.04	93.41 ± 0.90	0.23 ± 0.03
Fashion-MNIST	w/o all	89.93 ± 0.65	0.30 ± 0.02	89.15 ± 0.53	0.32 ± 0.02
	w/ cutout	87.77 ± 1.38	0.36 ± 0.04	88.69 ± 1.68	0.33 ± 0.05
	w/ repl.	93.66 ± 0.19	0.18 ± 0.01	93.12 ± 0.41	0.20 ± 0.01
	w/ repl. + cutout	92.63 ± 0.73	0.20 ± 0.02	93.68 ± 0.68	0.20 ± 0.01
SVHN	w/o all	92.38 ± 0.96	0.29 ± 0.03	88.10 ± 1.34	0.47 ± 0.04
	w/ cutout	79.21 ± 19.80	0.64 ± 0.57	82.60 ± 18.89	0.60 ± 0.53
	w/ repl.	96.33 ± 0.40	0.17 ± 0.01	93.94 ± 0.53	0.27 ± 0.01
	w/ repl. + cutout	93.38 ± 0.47	0.24 ± 0.02	94.43 ± 0.29	0.27 ± 0.02

A.1 Ablation study

We ran an ablation study to compare DensEMANN’s performance with and without the dense block replication mechanism described in Section 3.1.3, and with and without the use of cutout regularization [44] on the training and validation data. This experiment is mainly relevant for evaluating the performance of DensEMANN’s incremental approach *on its own*, i.e. without any interference from the block replication mechanism, when it comes to optimally training the generated one-block DenseNet.

For each setting in the ablation study (with and without block replication, with and without cutout), we ran the same experiments as in Sections 4.1 and 4.2: an evaluation of DensEMANN’s average performance over 5 runs, and a comparison of the best generated network’s performance with its original weights and when retrained 5 times from scratch. Aside from the ablation settings, we use the same experimental setup as in Section 4—which allows us to reuse the data in Table 1 for the setting with both replication and cutout.

A.1.1 DensEMANN’s full potential ablated (average performance, best networks)

This section corresponds to the experiment described in Section 4.1 of the main paper. Tables 3 and 4 show the results of running DensEMANN 5 times for each ablation setting and each benchmark task. Table 3 shows the average results of the five runs, and Table 4 shows the results of the “best” generated networks out of these five runs (i.e. the network that obtained the lowest validation loss).

Table 4: Ablation study on DensEMANN’s block replication and use of cutout: best generated DenseNet-BC after five runs on benchmark datasets

Dataset	DensEMANN setting	GPU execution time (hours)	GPU inference time (seconds)	Num. layers per block	Trainable parameters
CIFAR-10	w/o all	9.34 (39.75)	3.0	7	64778
	w/ cutout	10.38 (49.16)	2.8	6	58218
	w/ repl.	11.73 (51.68)	3.4	7	233418
	w/ repl. + cutout	16.55 (67.39)	3.5	7	245423
Fashion-MNIST	w/o all	3.20 (10.45)	2.7	2	13624
	w/ cutout	6.43 (19.24)	3.5	5	30392
	w/ repl.	4.85 (21.13)	3.8	2	50272
	w/ repl. + cutout	7.53 (32.75)	4.3	3	68638
SVHN	w/o all	1.73 (8.09)	11.2	11	92298
	w/ cutout	2.37 (8.75)	11.1	12	96676
	w/ repl.	2.76 (9.83)	13.1	11	358638
	w/ repl. + cutout	3.23 (16.96)	13.3	11	336066

Table 4: Ablation study on DensEMANN’s block replication and use of cutout: best generated DenseNet-BC after five runs on benchmark datasets, *continued*

Dataset	DensEMANN setting	Validation set		Test set	
		Acc. (%)	Loss	Acc. (%)	Loss
CIFAR-10	w/o all	88.12	0.37	87.31	0.40
	w/ cutout	83.08	0.49	85.77	0.44
	w/ repl.	92.90	0.20	92.73	0.23
	w/ repl. + cutout	91.34	0.26	93.91	0.21
Fashion-MNIST	w/o all	90.84	0.28	89.56	0.30
	w/ cutout	89.58	0.30	90.93	0.27
	w/ repl.	93.98	0.17	93.58	0.19
	w/ repl. + cutout	93.62	0.18	94.43	0.19
SVHN	w/o all	93.48	0.26	89.38	0.42
	w/ cutout	88.88	0.36	92.03	0.32
	w/ repl.	96.92	0.14	94.42	0.26
	w/ repl. + cutout	94.10	0.22	94.70	0.26

As in Table 1, for “best” networks we include two execution times: the one for the run that generated the network, and the sum of the execution times of all 5 runs (the time it took us to find and select this “best” network).

At first sight, these results seem to justify the use of the block replication (repl.) mechanism, as well as to some degree the use of cutout regularization. Both techniques appear to correlate with an increase in test set performance with respect to the “w/o all” results, although they also seem to correlate with an increase in size / parameter count and in GPU execution time.

To find the statistically significant differences (stat. sign. diff.) between the performances of each DensEMANN setting, we use two-way analyses of variance (ANOVAS), with Tukey’s range test as post-hoc analysis. The results of these tests are as follows:

- Both repl. and cutout correlate to significantly longer execution times. The P -values for CIFAR-10 were 7.72×10^{-4} for repl., 3.32×10^{-3} for cutout, and 0.40 for interaction. Similar P -values were obtained for Fashion-MNIST: 5.75×10^{-4} for repl., 2.35×10^{-3} for cutout, and 0.62 for interaction. For SVHN the situation is slightly different: 2.79×10^{-4} for repl., 2.29×10^{-3} for cutout, 8.25×10^{-3} for interaction. In that case Tukey’s test only shows stat. sign. diff. between the “w/ repl + cutout” experiment and all others.
- Inference times are significantly longer when using repl., but usually cutout does not affect them significantly. This is the case for CIFAR-10: $P = 4.60 \times 10^{-3}$ for repl., 0.95 for cutout, 0.62 for interaction. For SVHN the situation is similar: $P = 3.27 \times 10^{-5}$ for repl., 0.14 for cutout and 15.99×10^{-3} for interaction; Tukey’s test only shows stat. sign. diff.

691 between the two “w/ repl.” experiments and all others including each other. However, for
692 Fashion-MNIST there is a slightly stat. sign. diff. with cutout: $P = 5.78 \times 10^{-4}$ for repl.,
693 0.02 for cutout and 0.45 for interaction.

- 694 • Parameter counts are significantly higher when using repl., but cutout does not seem to
695 affect them significantly. This is the case for both CIFAR-10 ($P = 5.22 \times 10^{-7}$ for repl.,
696 0.86 for cutout, 0.72 for interaction) and Fashion-MNIST ($P = 5.17 \times 10^{-4}$ for repl., 0.14
697 for cutout, 0.37 for interaction). SVHN is an exception: $P = 4.39 \times 10^{-6}$ for repl., 0.06 for
698 cutout (almost stat. sign. diff.) and 0.04 for interaction. In that case Tukey’s test only shows
699 stat. sign. diff. between the two “w/ repl.” experiments and all others, including each other.
- 700 • The validation set performance (both the accuracy and the loss) significantly improves
701 when using repl., but worsens when using cutout. This is the case for both CIFAR-10 and
702 Fashion-MNIST: for the validation set loss,⁴ CIFAR-10’s P -values are 1.38×10^{-10} for
703 repl., 2.67×10^{-6} for cutout, and 0.03 for interaction (Tukey’s test shows stat. sign. diff.
704 between all experiments), while Fashion-MNIST’s are 1.64×10^{-9} for repl., 2.33×10^{-3}
705 for cutout, and 0.13 for interaction. SVHN is the exception, as for this benchmark there
706 were no stat. sign. diff. between the settings: $P = 0.06$ for repl., 0.12 for cutout and 0.31
707 for interaction. (There is an almost stat. sign. diff. for repl., and the accuracy is higher with
708 repl. than without it.)
- 709 • The test set performance significantly improves when using repl., but seems unaffected by
710 cutout. This is the case for all three datasets: for the test set loss, CIFAR-10’s P -values
711 are 8.14×10^{-12} for repl., 0.89 for cutout, and 0.033 for interaction (Tukey’s test only
712 shows stat. sign. diff. between the experiments with repl. and those without repl.); Fashion-
713 MNIST’s are 2.00×10^{-8} for repl., 0.56 for cutout, and 0.80 for interaction; SVHN’s are
714 0.04 for repl., 0.62 for cutout, and 0.62 for interaction.

715 We can thus conclude that the block replication method results in a significant improvement in
716 the network’s final performance, but at the expense of an also significant increase in the network’s
717 size and in DensEMANN’s GPU execution time. The use of cutout only clearly correlates with a
718 significant increase in the GPU execution time, but the observed increases in final performance and
719 network size are not statistically significant.

720 For the use of repl., the significant improvement in the network’s final performance is most likely
721 caused by the also significant increase in the network’s size: the more learnable weights a NN has,
722 the more complex are the target functions that it can represent. The origin of the increase in size is
723 obvious: the average sizes when using repl. are always (just over) 3 times the average sizes when not
724 using it, because the repl. method copies the DensEMANN-generated dense block $N = 3$ times (and
725 adds some transition layers). As for the increase in execution time, it most certainly corresponds to
726 the extra 300 epochs that take place after replicating the dense block. Indeed, the best networks in
727 Table 1 were also retrained during 300 epochs, and the mean execution times from that experiment
728 correspond very closely to the differences in execution time with and without repl. in Table 3 (for
729 instance, for CIFAR-10 the average retraining time in Table 1 is 3.86 hours, and the corresponding
730 increase in execution time is $13.48 - 9.83 = 3.65$ hours).

731 As for cutout, we hypothesise that the longer execution times are due to the extra time needed for
732 generating the batches of data-augmented images. The fact that cutout does not entail a significant
733 change in the inference time supports this hypothesis. For the non-significance of the changes in the
734 networks’ size and (especially) their test performance, we initially suspected a type II error caused by
735 small sample size and outliers in the data (there is in fact a very obvious outlier in the “SVHN w/
736 cutout” experiment, where DensEMANN generated a network with 12.92k parameters and a test set
737 accuracy of 48.89%). However, regardless of any outliers, the results with and without cutout are too
738 similar to rule out the null hypothesis.

739 It is also worth commenting that, in our workflow, when we use cutout we apply it on both the training
740 and validation data. The ANOVA results show that cutout has a significantly negative impact on the
741 validation set performance, which in our opinion is caused by the agresivity of this data augmentation
742 method (it removes a large random patch in each image). As explained before, there is no evidence of
743 a negative influence of cutout on the test set performance—but there is also no evidence of a positive

⁴We consider the ANOVA’s results for accuracy and loss to be interchangeable, as for each benchmark and for each data split (validation or test), the P -values for accuracies and losses are always very similar.

Table 5: Ablation study on DensEMANN’s block replication and use of cutout: retraining the best generated DenseNet-BC after five runs on benchmark datasets

Dataset	DensEMANN setting	Source of the weights	GPU execution time (hours)	GPU inference time (seconds)
CIFAR-10	w/o all	DensEMANN	9.34	2.97
		Retrain from scratch	2.53 ± 0.01	2.96 ± 0.01
	w/ cutout	DensEMANN	10.38	2.84
		Retrain from scratch	2.78 ± 0.00	2.83 ± 0.02
	w/ repl.	DensEMANN	11.73	3.45
		Retrain from scratch	3.38 ± 0.01	3.47 ± 0.02
Fashion-MNIST	w/o all	DensEMANN	16.55	3.47
		Retrain from scratch	3.86 ± 0.01	3.46 ± 0.04
	w/ cutout	DensEMANN	<i>3.20</i>	<i>2.75</i>
		Retrain from scratch	<i>1.59 ± 0.01</i>	<i>2.91 ± 0.12</i>
	w/ repl.	DensEMANN	<i>6.43</i>	<i>3.49</i>
		Retrain from scratch	<i>2.23 ± 0.13</i>	<i>2.84 ± 0.61</i>
SVHN	w/o all	DensEMANN	<i>4.85</i>	<i>3.79</i>
		Retrain from scratch	<i>2.93 ± 2.19</i>	<i>3.40 ± 0.14</i>
	w/ cutout	DensEMANN	<i>7.53</i>	<i>4.26</i>
		Retrain from scratch	<i>2.81 ± 0.02</i>	<i>3.75 ± 0.20</i>
	w/ repl.	DensEMANN	<i>1.73</i>	<i>11.24</i>
		Retrain from scratch	<i>0.63 ± 0.00</i>	<i>11.49 ± 0.08</i>
SVHN	w/o all	DensEMANN	<i>2.37</i>	<i>11.13</i>
		Retrain from scratch	<i>1.76 ± 2.16</i>	<i>11.60 ± 0.25</i>
	w/ cutout	DensEMANN	<i>2.76</i>	<i>13.08</i>
		Retrain from scratch	<i>0.88 ± 0.00</i>	<i>12.38 ± 0.23</i>
	w/ repl.	DensEMANN	<i>3.23</i>	<i>13.28</i>
		Retrain from scratch	<i>1.04 ± 0.17</i>	<i>11.76 ± 2.69</i>

influence on it. Further experiments are needed for testing cutout’s effect if it is used on the training data but *not* on the validation data.

A.1.2 Retraining our best networks from scratch: DensEMANN vs. “perfect” NAS

This section corresponds to the experiment in Section 4.2 of the main paper. In Table 5, we copy the GPU times and validation and test set performances obtained when generating the best networks in Table 4, and we compare these results to those obtained after retraining each of these networks from scratch. We use the same training settings as in the DensEMANN runs that generated each network: those networks that were generated with cutout are also retrained with cutout, and those that were generated without it are also retrained without it.

To compare the validation and test set performance before and after retraining each network, we use one-sample t-tests. Their results are as follows:

- No repl. or cutout (“w/o all” setting):
 - Validation set accuracy: the P -values were 0.10 for CIFAR-10, 1.00 for Fashion-MNIST, 1.45×10^{-4} for SVHN. There are stat. sign. diff. only for SVHN, where retraining caused the performance to improve.
 - Validation set loss: the P -values were 0.02 for CIFAR-10, 0.10 for Fashion-MNIST, 1.46×10^{-3} for SVHN. There are stat. sign. diff. for CIFAR-10 and SVHN, where retraining caused the performance to improve (in the case of CIFAR-10 only slightly).
 - Test set accuracy: the P -values were 0.05 (rounded up) for CIFAR-10, 0.01 for Fashion-MNIST, 1.31×10^{-5} for SVHN. There are stat. sign. diff. for all cases, where retraining caused the performance to improve (in the case of CIFAR-10 only slightly).
 - Test set loss: the P -values were 0.01 for CIFAR-10, 3.74×10^{-3} for Fashion-MNIST, 1.40×10^{-3} for SVHN. There are stat. sign. diff. for all cases, where retraining caused the performance to improve.
- Cutout, but no repl. (“w/ cutout.” setting):

Table 5: Ablation study on DensEMANN’s block replication and use of cutout: retraining the best generated DenseNet-BC after five runs on benchmark datasets, *continued*

Dataset	DensEMANN setting	Source of the weights	Validation set		Test set	
			Acc. (%)	Loss	Acc. (%)	Loss
CIFAR-10	w/o all	DensEMANN	88.12	0.37	87.31	0.40
		Retrain from scratch	87.94 \pm 0.19	0.36 \pm 0.01	87.69 \pm 0.31	0.38 \pm 0.01
	w/ cutout	DensEMANN	83.08	0.49	85.77	0.44
		Retrain from scratch	84.51 \pm 0.45	0.46 \pm 0.01	87.71 \pm 0.20	0.40 \pm 0.01
	w/ repl.	DensEMANN	92.90	0.20	92.73	0.23
		Retrain from scratch	92.91 \pm 0.18	0.22 \pm 0.00	92.87 \pm 0.17	0.23 \pm 0.01
	w/ repl. + cutout	DensEMANN	91.34	0.26	93.91	0.21
		Retrain from scratch	91.90 \pm 0.42	0.25 \pm 0.01	94.25 \pm 0.16	0.20 \pm 0.01
Fashion-MNIST	w/o all	DensEMANN	90.84	0.28	89.56	0.30
		Retrain from scratch	90.84 \pm 0.44	0.26 \pm 0.01	90.22 \pm 0.32	0.28 \pm 0.01
	w/ cutout	DensEMANN	89.58	0.30	90.93	0.27
		Retrain from scratch	91.41 \pm 0.66	0.24 \pm 0.02	92.32 \pm 0.16	0.24 \pm 0.01
	w/ repl.	DensEMANN	93.98	0.17	93.58	0.19
		Retrain from scratch	93.94 \pm 0.34	0.17 \pm 0.01	93.60 \pm 0.20	0.18 \pm 0.00
	w/ repl. + cutout	DensEMANN	93.62	0.18	94.43	0.19
		Retrain from scratch	93.70 \pm 0.53	0.18 \pm 0.01	94.47 \pm 0.22	0.19 \pm 0.01
SVHN	w/o all	DensEMANN	93.48	0.26	89.38	0.42
		Retrain from scratch	95.53 \pm 0.32	0.21 \pm 0.02	92.03 \pm 0.23	0.36 \pm 0.02
	w/ cutout	DensEMANN	88.88	0.36	92.03	0.32
		Retrain from scratch	91.83 \pm 0.23	0.28 \pm 0.01	93.23 \pm 0.14	0.31 \pm 0.02
	w/ repl.	DensEMANN	96.92	0.14	94.42	0.26
		Retrain from scratch	96.56 \pm 0.24	0.16 \pm 0.01	93.97 \pm 0.40	0.29 \pm 0.02
	w/ repl. + cutout	DensEMANN	94.10	0.22	94.70	0.26
		Retrain from scratch	93.81 \pm 0.39	0.23 \pm 0.01	94.50 \pm 0.16	0.26 \pm 0.01

- Validation set accuracy: the P -values were 2.10×10^{-3} for CIFAR-10, 3.51×10^{-3} for Fashion-MNIST, 9.24×10^{-6} for SVHN. There are stat. sign. diff. for all cases, where retraining caused the performance to improve.
- Validation set loss: the P -values were 5.22×10^{-3} for CIFAR-10, 2.23×10^{-3} for Fashion-MNIST, 2.40×10^{-5} for SVHN. There are stat. sign. diff. for all cases, where retraining caused the performance to improve.
- Test set accuracy: the P -values were 2.89×10^{-5} for CIFAR-10, 4.48×10^{-5} for Fashion-MNIST, 3.89×10^{-5} for SVHN. There are stat. sign. diff. for all cases, where retraining caused the performance to improve (in the case of SVHN only slightly).
- Test set loss: the P -values were 3.71×10^{-4} for CIFAR-10, 7.51×10^{-4} for Fashion-MNIST, 0.24 for SVHN. There are stat. sign. diff. for CIFAR-10 and Fashion-MNIST, where retraining caused the performance to improve.
- Repl., but no cutout (“w/ repl.” setting):
 - Validation set accuracy: the P -values were 0.93 for CIFAR-10, 0.78 for Fashion-MNIST, 0.03 for SVHN. There are stat. sign. diff. only for SVHN, where retraining caused the performance to *worsen* (albeit only slightly).
 - Validation set loss: the P -values were 4.54×10^{-3} for CIFAR-10, 0.40 for Fashion-MNIST, 0.08 for SVHN. There are stat. sign. diff. only for CIFAR-10, where retraining caused the performance to *worsen*.
 - Test set accuracy: the P -values were 0.14 for CIFAR-10, 0.87 for Fashion-MNIST, 0.07 for SVHN. There are no stat. sign. diff. (almost stat. sign. diff. for SVHN, where retraining caused the performance to slightly *worsen*).
 - Test set loss: the P -values were 0.98 for CIFAR-10, 0.21 for Fashion-MNIST, 0.03 for SVHN. There stat. sign. diff. only for SVHN, where retraining caused the performance to *worsen*.
- Both repl. and cutout (“w/ repl. + cutout” setting):
 - Validation set accuracy: the P -values were 0.04 for CIFAR-10, 0.75 for Fashion-MNIST, 0.17 for SVHN. There stat. sign. diff. only for CIFAR-10, where retraining caused the performance to improve (albeit only slightly).

- 798 – Validation set loss: the P -values were 0.09 for CIFAR-10, 0.60 for Fashion-MNIST,
799 0.52 for SVHN. There are no stat. sign. diff.
- 800 – Test set accuracy: the P -values were 0.01 for CIFAR-10, 0.68 for Fashion-MNIST,
801 0.04 for SVHN. There are stat. sign. diff. for CIFAR-10 and SVHN: for CIFAR-10
802 retraining caused the performance to improve (albeit only slightly) while for SVHN it
803 caused the performance to worsen (also slightly).
- 804 – Test set loss: the P -values were 0.06 for CIFAR-10, 0.33 for Fashion-MNIST, 0.90 for
805 SVHN. There are no stat. sign. diff. (almost stat. sign. diff. for CIFAR-10, where
806 retraining caused the performance to slightly improve).

807 In conclusion, when repl. is not used (i.e. when using only DensEMANN’s basic incremental
808 workflow) the original NN’s performance improves significantly after retraining, especially if cutout
809 is used. In contrast, when repl. is used the original NN’s performance is optimal: retraining does not
810 improve it significantly (at least not the loss value), and in some cases it can even make it worse.

811 We hypothesise that the main cause of the optimal weights obtained with repl. is the “best model
812 saving” approach that it incorporates, and that is also used for retraining the networks (see Sections
813 3.1.3 and 4.2). Indeed, repl. uses this approach just after replicating the generated dense block $N - 1$
814 times, which means that it is responsible for the full training of over $N-1/N = 2/3$ of the network. For
815 this reason, we advise further incorporating this “best model saving” approach within DensEMANN’s
816 training and growing workflow (see the future research lines suggested in Section 5).

817 Concerning GPU times, as explained in Section 4.2 the execution times for DensEMANN are always
818 significantly longer than those for retraining the networks. This means that DensEMANN is far from
819 “perfection” concerning GPU times (in Sections 2 and 4.2 we argue that the time needed to (re)train
820 the final candidate architecture is a good stand-in for a “perfect” execution time). The ablation study
821 also reveals that repl.’s extra 300 training epochs are not the only cause for DensEMANN’s long
822 execution times: even when not replicating the last block, DensEMANN takes longer to run than the
823 standard retrain. As evidenced in Section 4.3, using a different weight initialization method for new
824 components may speed up DensEMANN considerably—which is why we also suggest this as future
825 research in Section 5.

826 A.2 Naive NAS baseline

827 One could claim that DensEMANN is very long and complicated, and yet its search space is
828 very limited. If a naive NAS procedure explored a simplified version of DensEMANN’s search
829 space by fully training and testing each candidate architecture from scratch, could it reach similar
830 performance in less time? To assess this question, we coded a naive NAS algorithm that systematically
831 trains and tests increasingly larger DenseNet architectures out of an extremely simplified version of
832 DensEMANN’s search space.

833 All candidate architectures have in common: the same number N of dense blocks, the same number
834 k of convolution filters in each dense layer (for DenseNet-BC, the first convolution has got $4 * k$
835 filters, and the second one k filters), and that all of their dense blocks have got the same number of
836 layers l . The algorithm begins with $l = 1$ (N and k are user-set), and progressively tests networks
837 with increasingly large values for l . To this aim, it trains each candidate network for 300 epochs and
838 then evaluates its performance on the validation set. We use the same training methodology as in our
839 retraining experiments, but without cutout regularization.

840 The stopping criterion is similar to that of DensEMANN’s macro algorithm: if the current network’s
841 validation set accuracy does not improve at least $IT = 0.01$ beyond the previous network’s accuracy,
842 we select the previous network as the final candidate. There is also an optional stopping criterion
843 that sets a bound on the final candidate’s size: if increasing l would make the network’s number of
844 parameters surpass a certain *maximum parameter count* (MPC), then the current network becomes
845 the final candidate. Setting MPC to a none value removes this stopping criterion.

846 We ran the naive NAS algorithm 5 times for the CIFAR-10 benchmark, with either $N = 3$ or 1, and
847 without imposing the size-based stopping criterion. During each run, we recorded the details for
848 each candidate architecture (training and inference time, number of layers, number of parameters
849 and performance measures), as well as the execution time until it was evaluated. This allowed us to
850 simulate the use of the size-based stopping criterion for any MPC value, by considering, for each

Table 6: Naive NAS baseline for DenseNet-BC: average performance on CIFAR-10

Search space	GPU execution time (hours)	GPU final candidate training time (hours)	GPU inference time (seconds)	Num. layers per block	Trainable parameters (k)
$N = 3, MPC = \text{None}$	21.03 ± 2.77	3.36 ± 0.24	3.35 ± 0.10	7.0 ± 0.7	219.22 ± 31.29
$N = 3, MPC = 200k$	13.90 ± 0.06	3.01 ± 0.02	3.16 ± 0.03	6.0 ± 0.0	176.12 ± 0.00
$N = 1, MPC = \text{None}$	<i>25.42 \pm 4.98</i>	<i>3.26 \pm 0.37</i>	<i>3.96 \pm 0.43</i>	9.0 ± 1.6	82.76 ± 18.55
$N = 1, MPC = 60k$	<i>15.70 \pm 1.35</i>	<i>2.77 \pm 0.31</i>	<i>3.70 \pm 0.48</i>	7.0 ± 0.0	59.91 ± 0.00

Table 6: Naive NAS baseline for DenseNet-BC: average performance on CIFAR-10, *continued*

Search space	Validation set		Test set	
	Acc. (%)	Loss	Acc. (%)	Loss
$N = 3, MPC = \text{None}$	93.51 ± 0.29	0.20 ± 0.01	92.86 ± 0.36	0.23 ± 0.01
$N = 3, MPC = 200k$	92.78 ± 0.35	0.22 ± 0.01	92.25 ± 0.10	0.25 ± 0.01
$N = 1, MPC = \text{None}$	89.48 ± 1.10	0.32 ± 0.04	88.61 ± 1.15	0.35 ± 0.03
$N = 1, MPC = 60k$	87.68 ± 0.41	0.37 ± 0.01	87.34 ± 0.09	0.39 ± 0.00

run, the largest candidate that fulfilled the criterion as the final candidate. In all of our runs, we used $k = 12$ and $IT = 0.01$.

Table 6 shows the average results for this experiment. The MPC values for (simulated) size-based stopping were chosen to make the algorithm’s final candidate networks have similar sizes to the DensEMANN-generated networks. The GPU times for $N = 1$ are reported in *italized blue-gray* because they are highly unreliable: we ran one experiment in the MSi GT76, one in the MSi GT75, and three in the internal cluster. Those for $N = 3$ were all obtained on the MSi GT76.

We compare the results in table 6 to those in Table 3 for experiments without cutout.

Concerning the execution times, it is clear that DensEMANN outperforms the naive NAS algorithm. This is true both when the full runs are considered, and when size-based stopping is simulated with the selected MPC values.

As for the performance of naive NAS, we ran various statistical tests to compare it to DensEMANN’s. These were all two-sample T-tests except for parameter counts with simulated size-based stopping, for which we used one-sample T-tests because we always obtained null variances for naive NAS (the final candidate architecture was always the same):

- Size / parameter count: there are no stat. sign. diff. between the full naive NAS runs and DensEMANN’s for $N = 3$ ($P = 0.11$), but there are stat. sign. diff. between them for $N = 1$ ($P = 0.02$). We also confirm that the selected MPC values make the naive NAS algorithm’s architectures statistically similarly sized to DensEMANN’s architectures: for $N = 3$ the P -value is 0.93, and for $N = 1$ the P -value is 0.35.
- NN performance (both validation and test set): for both $N = 3$ and $N = 1$ there are always stat. sign. diff. between the full naive NAS runs and DensEMANN’s, but never between the simulated size-based stopping runs and DensEMANN’s:
 - Validation set accuracy: for $N = 3$ the P -values are 2.55×10^{-3} for full naive NAS, and 0.10 for simulated size-based stopping; for $N = 1$ they are 7.21×10^{-3} for full naive NAS, and 0.18 for simulated size-based stopping.
 - Validation set loss: for $N = 3$ the P -values are 0.01 for full naive NAS, and 0.25 for simulated size-based stopping; for $N = 1$ they are 6.43×10^{-3} for full naive NAS, and 0.10 for simulated size-based stopping.
 - Test set accuracy: for $N = 3$ the P -values are 0.04 for full naive NAS, and 0.65 for simulated size-based stopping; for $N = 1$ they are 0.01 for full naive NAS, and 0.09 for simulated size-based stopping.
 - Test set loss: for $N = 3$ the P -values are 0.03 for full naive NAS, and 0.45 for simulated size-based stopping; for $N = 1$ they are 6.69×10^{-3} for full naive NAS, and 0.07 for simulated size-based stopping.

886 In conclusion, the answer to our question is **no**, a naive NAS algorithm would not reach a similar
887 performance to DensEMANN’s in less time. Our naive NAS benchmark can reach statistically
888 identical accuracy levels to DensEMANN’s for similarly sized networks (and even higher accuracy
889 levels when building 3-block networks), but it always does so in significantly more time.

890 This said, it is worrying that our naive NAS baseline *does* reach DensEMANN’s same performance
891 level within a reasonable time frame (less than a GPU day). This indicates that this method is likely
892 capable of outperforming DensEMANN if it uses a quick and reliable performance estimator—not to
893 mention a zero-cost estimator [30, 31, 2]. Furthermore, the final candidate networks’ training times,
894 which as explained in other sections can be used to simulate the algorithm’s execution times if it used
895 a “perfect” performance estimator, are much lower than DensEMANN’s execution times. For these
896 reasons, we suggest designing a new baseline consisting of a zero-cost algorithm that explores the
897 same search space as DensEMANN (see the future research lines in Section 5).

898 **A.3 Full comparison against the state of the art**

899 We decided to extend Section 4.3’s CIFAR-10 state of the art to also include the experiments and
900 results reported in this appendix. To this aim, we created Table 7 and Figure 3 as extended versions
901 of (respectively) Table 2 and Figure 2 from that section.

902 From the distribution of DenseNet- and DensEMANN-related architectures in Table 7, it seems
903 that these architectures are limited by their own suboptimal size vs. error rate Pareto front. This
904 performance boundary may be caused by intrinsic performance limitations of DenseNet architectures—
905 at least as defined in [17]—or of DensEMANN’s search space. To test this hypothesis, we suggest
906 widening DensEMANN’s search space explore to less standard “DenseNet-like” architecture designs.
907 For instance, the macro-algorithm could have a wider range of choices for connecting layers inside
908 a dense block, or for deciding when to stop growing the current block and start a new one (see our
909 suggested future research lines in Section 5).

Table 7: Performance comparison of DensEMANN and the naive NAS baseline (various configurations) against human-designed NN models and state-of-the-art NAS algorithms, for architectures with less than 10 million parameters

Category	Name	Trainable parameters (M)	Error rate on CIFAR-10 (%)	GPU execution time (days)
Human-designed	ResNet 20 [48]	0.27	8.75	N/A
	ResNet 110 (as reported by He et al. [48])	1.7	6.61 ± 0.16	N/A
	ResNet 110 (as reported by Huang et al. [49])	1.7	6.41	N/A
	ResNet 110 with Stochastic Depth [49]	1.7	5.25	N/A
	WRN 40-1 (no data augmentation) [50]	0.6	6.85	N/A
	DenseNet 40 ($k = 12$) [17]	1	5.24	N/A
	DenseNet-BC 100 ($k = 12$) [17]	0.8	4.51	N/A
	Highway 1 (Fitnet 1) [51]	0.2	10.82	N/A
	Highway 4 [51]	1.25	9.66	N/A
	Petridish initial model ($N = 6$, $F = 32$) + cutout [13]	0.4	4.6	N/A
Reinforcement learning (RL)	NAS-RL / REINFORCE (v1 no stride or pooling) [52]	4.2	5.5	22400
	NAS-RL / REINFORCE (v2 predicting strides) [52]	2.5	6.01	22400
	NAS-RL / REINFORCE (v3 max pooling) [52]	7.1	4.47	22400
	NASNet-A (6 @ 768) [36]	3.3	3.41	2000
	NASNet-A (6 @ 768) + cutout [36]	3.3	2.65	2000
	Block-QNN-S, $N = 2$ [53]	6.1	4.38	96
Evolutionary and genetic algorithms (EA and GA)	Large-Scale Evolution [54]	5.4	5.4	2600
	CGP-CNN (ConvNet) [55]	1.5	5.8	12
	CGP-CNN (ResNet) [55]	1.68	5.98	14.9
	AmoebaNet-A ($N = 6$, $F = 32$) [56]	2.6	3.4 ± 0.08	3150
	AmoebaNet-A ($N = 6$, $F = 36$) [56]	3.2	3.34 ± 0.06	3150
	EcoNAS + cutout [57]	2.9	2.62 ± 0.02	8
Gradient-based optimization (GO)	ENAS + micro search space [58]	4.6	3.54	0.45
	ENAS + micro search space + cutout [58]	4.6	2.89	0.45
	DARTS (1st order) + cutout [59]	3.3	3 ± 0.14	1.5
	DARTS (2nd order) + cutout [59]	3.3	2.76 ± 0.09	4
	XNAS-Small + cutout [60]	3.7	1.81	0.3
	XNAS-Medium + cutout [60]	5.6	1.73	0.3
	XNAS-Large + cutout [60]	7.2	1.6	0.3
Growing / Forward NAS	NASH ($n_{steps} = 5$, $n_{neigh} = 8$, 10 runs) [14]	5.7	5.7 ± 0.35	0.5
	LEMONADE SS-I + mixup + cutout [19]	0.047–3.4	8.9–3.6	80
	LEMONADE SS-II + mixup + cutout [19]	0.5–13.1	4.57–2.58	80
	Petridish macro ($N = 6$, $F = 32$) + cutout [13]	2.2	2.85 ± 0.12	5
	Petridish cell ($N = 6$, $F = 32$) + cutout [13]	2.5	2.87 ± 0.13	5
	Petridish cell, more filters ($N = 6$, $F = 37$) + cutout [13]	3.2	2.75 ± 0.21	5
	Firefly, WRN 28-1 seed + BN [10]	4	7.1 ± 0.1	N/A
	GradMax, WRN 28-1 seed + BN [10]	4	7.0 ± 0.1	N/A
Random Search	DARTS Random + cutout [59]	3.2	3.29 ± 0.15	4
	NASH Random ($n_{steps} = 5$, $n_{neigh} = 1$) [14]	4.4	6.5 ± 0.76	0.19
	LEMONADE SS-I Random + mixup + cutout [19]	0.048–2	10–4.4	80
Ours (DensEMANN)	DensEMANN (w/o repl.)	0.056 ± 0.009	13.91 ± 1.28	0.33 ± 0.05
	DensEMANN (w/o repl.) + cutout	0.053 ± 0.006	14.67 ± 0.71	0.41 ± 0.02
	DensEMANN (w/ repl.)	0.178 ± 0.042	7.87 ± 0.56	0.43 ± 0.05
	DensEMANN (w/ repl.) + cutout	0.186 ± 0.057	6.59 ± 0.90	0.56 ± 0.11
Ours (DensEMANN best)	DensEMANN (w/o repl.)	0.065	12.69	1.66
	DensEMANN (w/o repl.) + cutout	0.058	14.23	2.05
	DensEMANN (w/ repl.)	0.233	7.27	2.15
	DensEMANN (w/ repl.) + cutout	0.245	6.09	2.81
Ours (DensEMANN best retrained)	DensEMANN (w/o repl.)	0.065	12.31 ± 0.31	1.76 ± 0.00
	DensEMANN (w/o repl.) + cutout	0.058	12.29 ± 0.20	2.16 ± 0.00
	DensEMANN (w/ repl.)	0.233	7.13 ± 0.17	2.29 ± 0.00
	DensEMANN (w/ repl.) + cutout	0.245	5.75 ± 0.16	2.97 ± 0.00
Ours (Naive NAS)	Naive NAS baseline, $N = 1$, $MPC = \text{None}$	1.06 ± 0.21	11.39 ± 1.15	82.76 ± 18.55
	Naive NAS baseline, $N = 1$, $MPC = 60k$	0.65 ± 0.06	12.66 ± 0.09	59.91 ± 0.00
	Naive NAS baseline, $N = 3$, $MPC = \text{None}$	0.88 ± 0.12	7.14 ± 0.36	219.22 ± 31.29
	Naive NAS baseline, $N = 3$, $MPC = 200k$	0.58 ± 0.00	7.75 ± 0.10	176.12 ± 0.00

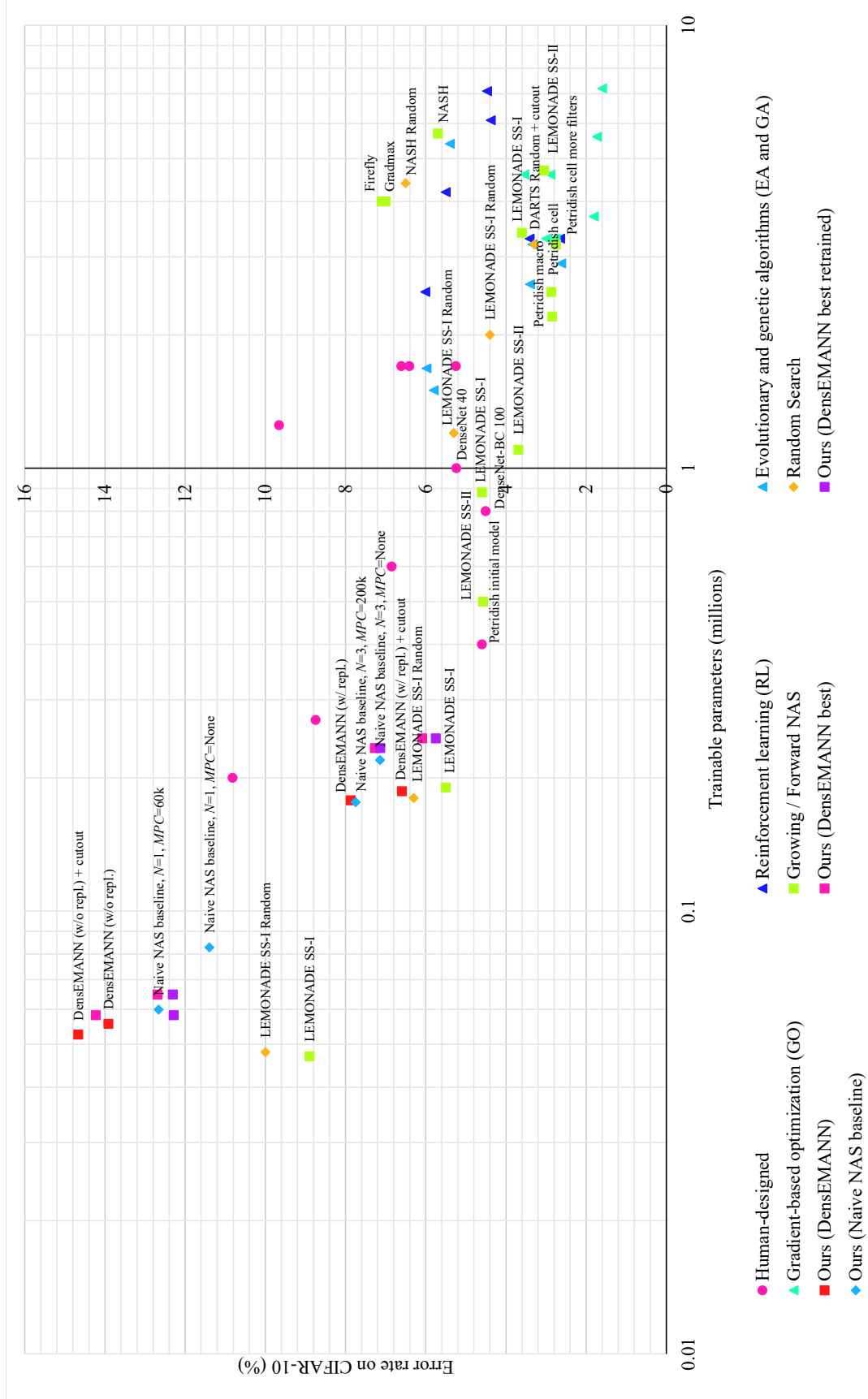


Figure 3: Scatter plot of the performance of the human-designed NN models and NAS algorithms in Table 7 (including DenseEMANN and the naive NAS baseline): accuracy on CIFAR-10 vs. size in trainable parameters. Labels are included for all algorithms in the “Growing / Forward NAS”, “Random Search”, “Ours (DenseEMANN)” and “Ours (Naive NAS)” groups, as well as for some human-designed NN (DenseNets and the Petridish initial model—which is on the Pareto front).