Self-Optimisation of Dense Neural Network Architectures: An Incremental Approach

Antonio García Díaz
IRIDIA-CoDE
Université libre de Bruxelles (ULB)
Brussels, Belgium
agarciad@ulb.ac.be

Hugues Bersini
IRIDIA-CoDE
Université libre de Bruxelles (ULB)
Brussels, Belgium
bersini@ulb.ac.be

***Abstract*—This paper presents a newly developed self-structuring algorithm for generating convolutional neural networks, as well as the results of preliminary tests performed on it. The algorithm produces DenseNet and DenseNet-BC architectures layer by layer from scratch, at the same time as they are being trained. Experimental results for well-known image classification datasets (CIFAR-10 and SVHN) are promising. The accuracy levels of generated networks are not significantly different than those of prebuilt DenseNet and DenseNet-BC with similar topologies, and are approaching the state of the art for these datasets.**

Keywords—neural network, architecture, topology, optimization, self-structuring, DenseNet, EMANN, connection strength

I. Introduction

An important aspect of the construction of a neural network (NN) model is the architecture, or topology, of the network – that is, its inner structure. Indeed, the so-called "topology problem" is one of the main concerns when implementing NN: how many layers, how many neurons (or kernels) per layer, and what connection scheme for these neurons and layers, would make the NN better suited for a given task.

The aim of the topology problem is to make a NN perform better on a task, by overcoming limitations that are not due to the learning algorithm, but rather inherent to the selected topology and to some of its settings [1]. Nevertheless, parameters such as the learning rate, momentum value, and epoch size for the learning scheme may also be considered part of the topology problem [2].

One of the most obvious aspects of the topology problem is the choice of the size of the NN architecture. On one hand, a poorly structured NN [3] or a NN with few hidden neurons [4] may underfit the dataset, as it will lack the representational power to model the diversity of the dataset's items, or the complexity inherent in them. On the other hand, an architecture that has been structured so as to suit every single item in the dataset may cause the system to be overfitted [5], owing to its excess information capability [6].

Nowadays the topology problem is often solved through means of trial and error [4]: candidate topologies are handcrafted by humans, and their parameters' values are tuned through experimentation and testing. This process often becomes a tedious and time-consuming search for the right values [5]. For this reason, there is growing interest in techniques for automatically designing NN architectures [7], despite these techniques not being widely used today (as they can be computationally expensive).

In order to become a viable alternative, these algorithms should produce topology designs that are both more efficient for a given task and simpler in their design than their handcrafted equivalents are. By "more efficient for a given task", it is understood that the architecture, once properly trained, can produce predictions for a given problem that are as accurate as possible. By "simpler in their design", it is understood that the topology has a small and clean design scheme: as few layers as possible, and very few (if any) loops and branches.

This paper presents the preliminary results obtained with a newly developed algorithm of the kind described above. This is a self-structuring algorithm – it builds a convolutional neural network (CNN) architecture from scratch, by incrementally adding layers on top of each other while the network is being trained, and stops when a point that is deemed sufficiently "optimal" is reached.

The architectures produced by this newly developed algorithm are based on the DenseNet and DenseNet-BC Densely Connected Convolutional Networks [8]. DenseNet are a kind of CNN architectures characterized by their "dense blocks", network sections where convolutional layers are made explicitly complementary to each other (Fig. 1). The input of each layer is a concatenation of all the previous outputs in the block, together with the block's global input. DenseNet-BC are optimized DenseNet where the input size of each layer is reduced. This is done by means of bottleneck layers at each layer output and a compression mechanism at the end of each block. The DenseNet and DenseNet-BC architectures built by the self-structuring algorithm presented in this paper only contain one such dense block, where new layers are stacked at the output of the previous ones at certain events during the network's training.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

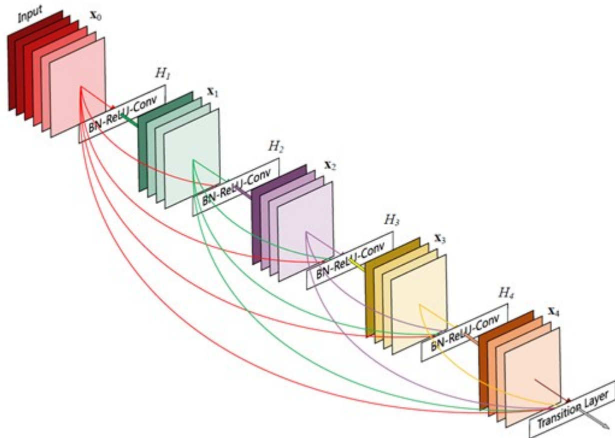

Fig. 1. Diagram showing the concatenations of layer outputs (including the global input) in a dense block. Each layer's output feature maps are represented in a different colour. Taken from [8].

II. Related Work

Algorithms where the NN is self-structured (or self-constructed) dynamically, while it is being trained, provide a timely solution for displacing some of the burden from system designers. It is for this reason that they have gained the attention of many researchers [4]. For instance, in 1994 Sam Waugh [1] reviewed various strategies for gradually improving NN architectures by means of pruning: pruning on connections, pruning on weights, and construction or pruning of hidden nodes. More recently (in 2016), Fadi Thabtah, Rami M. Mohammad, and Lee McCluskey [4] presented an opposite solution where the NN model is restructured by incremental means: tuning some parameters, adding new neurons to the hidden layer or sometimes adding a new layer to the network

A. Different Approaches to Self-Structuring

Three main approaches exist to automating a dynamic self-structuring process for a NN [9]. These are constructive algorithms, pruning algorithms, and constructive-pruning algorithms.

Constructive Algorithms (also known as incremental algorithms) start with a simple NN architecture, such as one hidden layered NN with a single neuron in the hidden layer [6]. Recursively, new items and parameters (hidden layers, hidden neurons, connections, etc.) are added to the initial topology, until a satisfactory result is reached. After each addition, either the entire network architecture or only the recently added parameter(s) are retrained, and they cannot be modified. Constructive algorithms tend to be relatively simple to tune because they usually depend on few initial parameters. These algorithms are also computationally efficient, since they search for small structures [4].

The first constructive algorithm likely was Dynamic Node Creation (DNC) [10]. This is a simple algorithm: neurons are added to the hidden layer one by one whenever the mean square error on the training dataset stops variating, until a predefined desired error rate is reached. Its main drawback is that, since each addition is irreversible, it tends to produce messy topologies [4]. Another example of a constructive algorithm is the Cascade-Correlation algorithm (CC-Alg.) [11], on which several improvements have been proposed, such as [12].

Pruning Algorithms start with an oversized NN architecture (i.e.: multiple hidden layers and many neurons in each layer) [4]. They work by recursively removing items and parameters (hidden layers, hidden neurons, connections, etc.) from the network until the final architecture is achieved. After each pruning phase, the new network is retrained for a while to let it compensate any post-pruning loss in the performance. However, if the performance does not improve, the deleted parameters are restored and others are removed instead. Usually, only one item or parameter is removed in each pruning phase [10]. This approach is not without disadvantages: it may be too time consuming, and the programmer needs to decide beforehand how big the initial NN topology should be for the specific problem in question.

Constructive-Pruning Algorithms work in two phases: a constructive phase and a pruning phase [4]. During the constructive phase new hidden layers, neurons, and connections are added. This phase may result in a ridiculously complicated topology, so a pruning phase is employed to simplify this topology while preserving the network's performance. Examples of constructive-pruning algorithms for NN can be found at [6], [13] and [14].

B. The EMANN Self-Structuring Algorithm

Particularly relevant for this paper is the EMANN algorithm [15]. EMANN is mostly a constructive approach, but it also sometimes uses pruning.

In this algorithm, the topology is built from "modules" or "blocks", which are progressively stacked on top of each other not unlike layers in the DenseNet CNN architecture. These "blocks" represent MLP networks with two layers each. They are connected together in a way that ensures complementary behavior (Fig. 2). Whenever a new MLP "block" is added, a constructive algorithm is performed on its first layer (or "hidden layer").

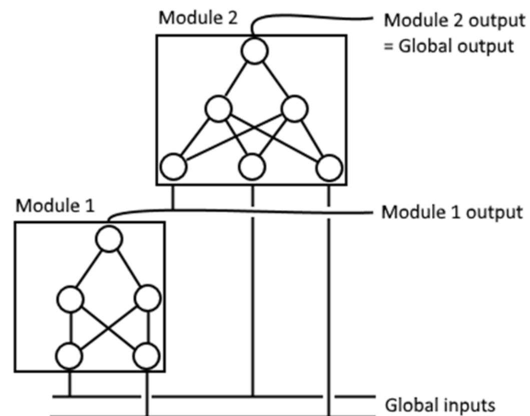

Fig. 2. Diagram showing an example of the topology structures generated by EMANN. "Modules" or "blocks" are stacked together in such a way as to ensure complementarity. Adapted from [15].

Neurons are added to this hidden layer (and sometimes pruned from it) based on the value of a feature called the connection strength (CS). For each neuron in a MLP's hidden layer, the CS is merely the average (mean) of the weights of all of its connections.

Because of the specific activation function used by EMANN neurons (the sigmoid), the CS proves to be a very good indicator of whether or not a neuron is extracting some feature from the data, and of how "settled" on identifying this feature the neuron has become. If the CS is very high, the neuron has clearly settled on identifying a feature, and if it is very low, the neuron is not successfully identifying anything useful.

III. METHODOLOGY

A. The Newly Developed Algorithm

The new self-structuring algorithm automatically constructs DenseNet and DenseNet-BC architectures from scratch while they are being trained. Specifically, the algorithm performs a training routine on what is, initially, an extremely simple DenseNet or DenseNet-BC network (one dense block with only one layer). Meanwhile, in-between certain epochs of the training process, the algorithm may modify the network by adding new layers at the end of its dense block. As a result, a network with a suitable layer count is successfully built and trained from scratch.

The main two reasons why DenseNet and DenseNet-BC were chosen as the basis for the new algorithm are that the performance level of these CNN topologies is very close to the state of the art for most well-known image classification datasets, and that some relevant parallels can be made between DenseNet and the topologies that EMANN produces. Furthermore, unlike other similar CNN architectures such as Residual Networks (ResNet) [16] and Crescendo Networks (CrescendoNet) [17], DenseNet make it possible to build a shallow but optimal scheme by progressively adding layers one by one. This makes little sense with CrescendoNet because of their path-based topology, and may take a very long time with ResNet due to their focus on depth.

The algorithm is based on a measurable feature in each layer of a DenseNet. This feature is called "connection strength" or CS. It derives its name from the connection strength used in the EMANN algorithm, to which it is analogous. There is one CS for every connection between any layer l and a previous layer s – $CS_{l,s}$ (the CS between l and s) is the mean (average) of the filter weights connecting l to s, taken in their absolute (positive) value.

In the new algorithm, CS values are compared together in order to extract a measurable feature that identifies layers as a whole, rather than connections between them. This has been called a "layerwise interpretation of CS", or LCS.

The LCS that is used in the algorithm has been named "relevance for sources". It is a fraction that expresses, for a given layer, how many of its sources treat that layer as a relevant output for sending information (Fig. 3).

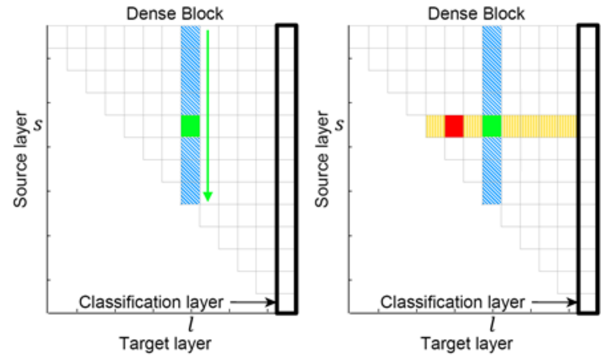

Fig. 3. Visualisation of the steps for calculating "relevance for sources" LCS. An iteration (solid green) is made through connections (slanted blue stripes) between a layer l and any of its source layers s. The CS of each of these connections is compared to the maximum CS (solid red) for the connections between source layer s and its destinations (vertical yellow stripes).

The "relevance for sources" for a given layer l is calculated through the following algorithm:

- For any given connection between layer l and a previous layer s:
  - If $CS_{l,s} \geq n * max_{\lambda}(CS_{\lambda,s})$, add '1' to the LCS ($n$ is such that $0 \leq n \leq 1$).
  - Else, add '0' to the LCS.
- Divide the final LCS value by the number of connections between l and any previous layer (normalization).

The value of n, the fraction of the maximum LCS for source layer s that $CS_{l,s}$ is compared with, is usually set to 0.67 (around two thirds).

The algorithm (Fig. 4) begins by initializing a DenseNet (or a DenseNet-BC) with a single dense block, and only one layer inside that block. The growth rate (the number of new convolutions per layer) is set to 12, and the network's weights are initialized at random. Afterwards, successive training epochs are performed on the DenseNet, with a learning rate initially fixed at 0.1.

Meanwhile, a self-structuring procedure is performed on the DenseNet. This procedure follows two successive stages: the so-called "ascension stage" and "improvement stage".

- The first of these stages is the "ascension stage", the main topology-building stage of the algorithm. It is a stage of quick growth for the network, and is guided by loops with a fixed duration. This duration, measured in training epochs, is a settable parameter called the "ascension threshold" or 'AT'. The loops mark the rhythm at which new layers are stacked at the end of the dense block: whenever a loop (which lasts for 'AT' epochs) ends, a new layer is added. The ascension stage normally ends when three or more layers have been stacked together, and one of the layers "settles" – that is, when its LCS reaches a value of 1.

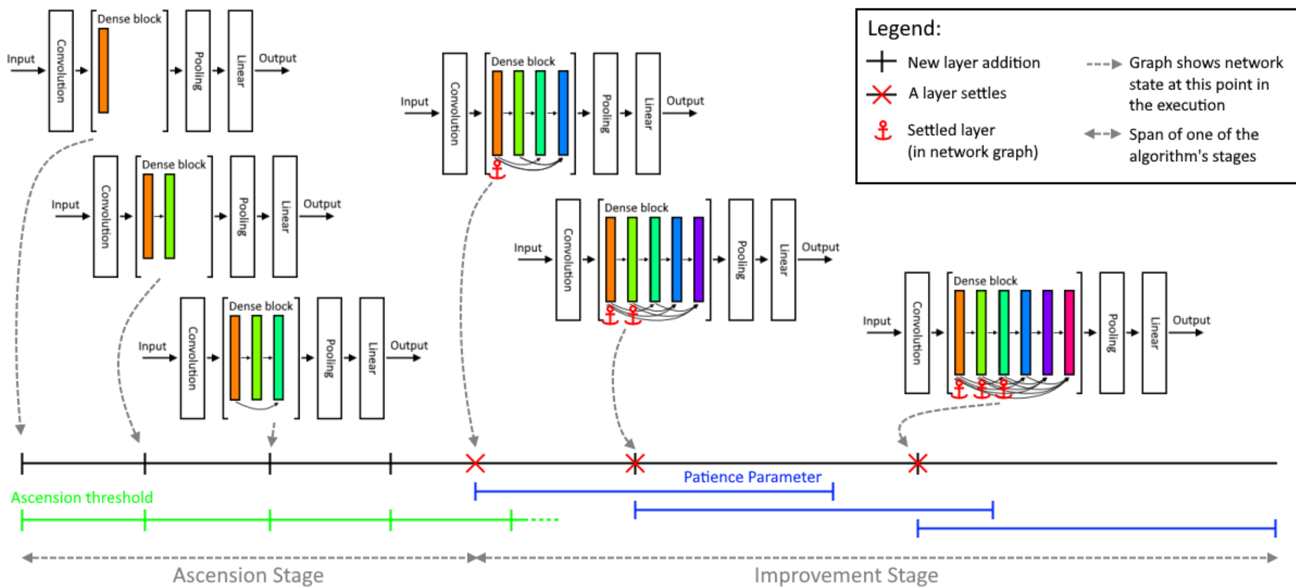

Fig. 4. Visualisation of a typical execution of the algorithm. Shows the evolution of a DenseNet's architecture throughout the two stages of the algorithm (ascension and improvement). During the "ascension stage", new layers are added to the dense block at the end of cycles with a length of 'AT' training epochs. Under normal conditions, the ascension stage lasts until there are at least three layers in the block and at least one of them has settled (in the figure's case there are four layers in the block when the first one settles). The next stage is the "improvement stage", which depends on a countdown of 'PP' epochs. Whenever another layer settles, a new layer is added to the dense block and the countdown is reset. The improvement stage ends when this countdown is able to end.

- In order to prevent the algorithm from getting stuck in the ascension stage (a phenomenon which has been empirically observed to occur with some datasets), this stage can be terminated if the accuracy of the network has not changed much in the latest epochs. The algorithm may thus exit the ascension stage if, in a window of the (usually 50) latest accuracies (measured on the validation set), the standard deviation is below a certain tolerance threshold (usually set to 0.1).
- After this comes the "improvement stage": a "waiting" stage where only a few layers are added at specific moments. Indeed, since sudden layer additions could disrupt the training process, a stage where the network remains mostly unchanged is necessary for it to recover and reach optimal accuracy levels. This stage is guided by a countdown system. The expected length of the countdown, in epochs, is yet another settable parameter called the "patience parameter" or 'PP'. When the improvement stage begins, a countdown with a duration of 'PP' training epochs starts, at the end of which both the training epochs and the algorithm end. Nevertheless, if any additional layer "settles", a new layer is stacked at the end of the block and the countdown starts again from the beginning.

During the training and self-structuring process, the learning rate may either remain constant at 0.1 or change under specific conditions. These changes in the learning rate are mostly reductions, and may be applied in either of two versions:

- In the first version, which will hereafter be called "reduce LR #0", the learning rate is divided by 10 two times during the 'PP' countdown: one when 50% of the countdown has elapsed, and another one at 75% of the countdown. In case the countdown is restarted, the learning rate returns to its original value of 0.1.
- In the second version, which will be called "reduce LR #1", the learning rate is also divided by 10 at the same points during the 'PP' countdown. However, these changes are now permanent, and can only be applied once. This means that the learning rate is only modified two times during the training: the first time that the countdown gets past 50% of the PP, and the first time that the countdown gets past 75% of the PP.

The previously described self-structuring algorithm was implemented in Python using the TensorFlow library. The DenseNet and DenseNet-BC implementation by Illarion Khlestov [18] was used as a basis for the algorithm's source code. An executable file, run_dense_net.py, is used for commanding the self-structuring, training and testing operations. The network itself is implemented as an object in the file models/dense_net.py. The most relevant functions for building and training networks are also found in that file. The code is available on GitHub [19].

B. Experimental Work for Designing the Algorithm

The analogous CS for the new self-structuring algorithm, as well as its layer-wise interpretation (the LCS), were developed on basis of a set of experiments. These had the objective of observing changes in the network's accuracy and some weight-related features when simple self-structuring routines were carried out. Meaningful correlations between these observations would then be established.

In the first of these experiments, it was intended to identify changes in DenseNet after layer additions, and to understand at which points it is "better" – with respect to performance –

to add a layer. A naïve routine of layer additions was performed on a simple DenseNet with only one block and (initially) one layer. During a training period of 80 epochs, layers would be added one by one every 10, 20, or 40 epochs. The additions would stop either after 40 epochs or at the end of the training.

Two kinds of measures were taken in this experiment: measures on the network's performance (its accuracy and cross-entropy loss, as well as a loss calculated at each layer as if it were the last layer in the block), and measures on each layer's learned parameters (the mean and standard deviation of the weights in each layer's convolutional filter). Comparisons were also made with measures on prebuilt DenseNet, containing only one block with 1, 2, or 4 layers.

The experiment showed that, in general, layer additions tend to make the accuracy grow, but also tend to move layer weights closer to 0. After comparing the experiment's results with data from the original DenseNet paper [8], it became apparent that the learning patterns of DenseNet would initially need to be measured for each connection independently.

Indeed, because of the design of DenseNet architectures, in increasingly deep networks some connections are given more "importance" than others. During training, the weights of less "important" connections get closer to 0 than other weights, limiting the impact of these connections through a "zeroing-out" effect. This could result in a smaller global mean (and standard deviation) for a layer's weights. This became the principle after which the analogous CS was modelled.

In the second experiment, it was intended to see if this CS could be generalized to a relevant feature defining an entire layer. The starting point of this experiment was a normalized version of the CS, where the CS between a layer and one of its sources is divided by the maximum CS in that layer.

The experiment consisted in measuring the evolution of normalized CS values for 300 epochs in two kinds of contexts. On one side, one-block DenseNets starting with one layer, where a naïve layer addition routine was applied (adding a layer every 40, 60, or 80 training epochs). On another, prebuilt DenseNets with either 1 very deep block (6, 12, or 18 layers) or 3 shallower blocks of equal depth (2, 4, 6, or 12 layers).

Results showed that, after each layer was added, its normalized CS values tended to rise until reaching a sort of equilibrium. Then, they plateaued until the end of the training. This equilibrium position tended to be closer to 1 (the max CS value) than 0 (a complete "zeroing-out"). However, in layers at the end of the block, normalized CS values tended to be more spread out at their equilibrium point, sometimes even falling under 0.5 (CS value closer to 0 than to the max CS).

By definition, the connections with the highest CS are also those with the highest weight values assigned to them. At the same time, higher weight values cause the information travelling through a connection to have more repercussion in the network's operations – more "importance". Since, as this second experiment shows, normalized CS values of early layers tend towards an equilibrium point close to 1, this means that all of their sources play important roles in their operations. A very likely cause is that these layers have learned to exploit all of their sources optimally, implying that the layers have been optimally trained.

In conclusion, to evaluate how well a DenseNet layer is trained, one could compare its CS values together, and see how many of them are high enough for the connection to be considered "important" or "relevant". In the new algorithm, this became the basis for the layer-wise interpretations of the CS, or LCS.

C. *Testing the Performance of the Algorithm*

The algorithm was tested by using it to produce both DenseNet and DenseNet-BC architectures. Initially, these architectures were trained on the CIFAR-10 dataset [20] in order to find optimal values for the 'AT' and 'PP' parameters empirically. Afterwards, tests with both the CIFAR-10 and SVHN [21] datasets were carried out, using both reduce LR #0 and reduce LR #1 and the following parameter values: growth rate = 12, (initial) learning rate = 0.1, AT = 10 epochs, and PP = 200 epochs (these values for 'AT' and 'PP' were previously found to produce optimal results).

The above-mentioned tests were also run for a second time using a version of the algorithm that did not rely on the CS to end its ascension stage (only stopping based on the validation set accuracy). Tests with a constant learning rate were also initially carried out, but these produced worse results than tests that used learning rate modification schedules did.

In addition, a reference test was devised that consisted in training and testing prebuilt topologies on the CIFAR-10 and SVHN datasets. These topologies were DenseNet or DenseNet-BC of the same kind as those produced by the algorithm (architectures with only one dense block that contained a fixed number of layers). The topologies were initialized with random weights, and then trained for 300 epochs. The learning rate was initialized at 0.1, and then divided by 10 at epochs 150 and 225 (50% and 75% of the training).

Each test (both the self-structuring and reference ones) was replicated five times with the same parameters. In order to assess the evolution of the accuracy over time, a random validation set was split off the training data at every epoch. At the end of each test, for the sake of comparison, the final accuracy was calculated on both the last generated validation set and the testing set. The final cross-entropy losses were also measured on these two datasets. These were stored together with the architecture's final number of layers and the total number of epochs performed. The mean and standard deviation of these values (over the five replicas) were then calculated to produce the final results.

The main objective of these tests was to compare the mean final accuracies obtained for each kind of test. It was hoped to generate self-structured DenseNet and DenseNet-BC that, compared with similar prebuilt structures and with the state of the art, would exhibit a higher accuracy for similar topologies or the same accuracy for smaller topologies. The best performing algorithm variants and parameter values for achieving this goal had to be identified, as well as means to improve them.

IV. Experimental Results

A. Results Obtained with CIFAR-10

The results obtained with CIFAR-10 can be found in Table I. Those obtained for their equivalent prebuilt networks, whose number of layers corresponds to the average layer count in each self-structuring test, can be found in Table II. Most of the values were rounded to the closest figure with two decimals. The only exceptions are the mean and standard deviation of the final number of epochs, which has been rounded to the closest integer.

In CIFAR-10, a validation set corresponds to 5,000 random examples cut off the full training set at each epoch, so that the network is not trained on the entire training set (with 50,000 examples) every time. The validation set mentioned in both tables is the one used in the last training epoch. In both tables also, the entire CIFAR-10 test set with 10,000 examples was used – the network does not process it until the end of the training.

Currently, the best mean accuracy values are above 80% on the test set. The low mean number of layers produced (around 6 or 7) makes these generated structures very simple, but it may in fact prevent them from reaching higher accuracy levels. The obtained results are nevertheless approaching the state of the art (99% accuracy for the CIFAR-10 test set) [22].

TABLE I. Self-Structuring Tests Using CIFAR-10

Self-Structuring Tests Using CIFAR-10			Test Results					
			Number of Layers	*End Epoch*	*Validation Set*		*Test Set*	
					Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*
Dense Net	Reduce LR #1	Mean	6.80	355	76.38	0.74	75.52	0.77
		STD	1.48	160	2.29	0.06	2.11	0.07
	Reduce LR #0	Mean	6.00	308	74.46	0.80	73.31	0.82
		STD	0.71	26	3.80	0.16	4.04	0.18
Dense Net-BC	Reduce LR #1	Mean	9.80	459	79.35	0.71	78.29	0.73
		STD	1.64	81	2.51	0.13	3.02	0.13
	Reduce LR #0	Mean	8.40	422	82.05	0.57	80.75	0.59
		STD	0.89	63	3.00	0.11	2.92	0.11

TABLE II. Prebuilt Architecture Tests Corresponding to Self-Structured Architectures, Using CIFAR-10

Prebuilt Architecture Tests Using CIFAR-10			Test Results			
			Validation Set		*Test Set*	
			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*
Dense Net	7 layers (Reduce LR #1)	Mean	77.25	0.72	76.68	0.74
		STD	2.13	0.11	2.48	0.12
	6 layers (Reduce LR #0)	Mean	77.84	0.69	76.78	0.71
		STD	3.03	0.09	3.43	0.10
Dense Net-BC	10 layers (Reduce LR #1)	Mean	84.56	0.57	83.74	0.58
		STD	1.90	0.10	2.44	0.11
	8 layers (Reduce LR #0)	Mean	77.27	0.86	76.45	0.86
		STD	1.71	0.11	2.02	0.12

The mean accuracies of the self-constructed networks obtained in these tests are also very close to those of prebuilt DenseNet and DenseNet-BC with architectures corresponding to their mean number of layers (rounded to the closest integer).

The version of the algorithm that used reduce LR #1 shows the best mean accuracies on the CIFAR-10 test set for DenseNet: 75.52% (vs. 76.68% mean accuracy for prebuilt DenseNet with 7 layers). For DenseNet-BC, it is the version that uses reduce LR #0 that results in a greater accuracy: 80.75% (vs. 83.74% for prebuilt DenseNet-BC with 10 layers).

In the case of DenseNet, an analysis of variance (ANOVA) showed that no significant difference existed between the mean accuracies of self-constructed and prebuilt examples. The analysis resulted in a 0.4514 p-value between the topologies generated with reduce LR #1 and corresponding prebuilt topologies, and a 0.1810 p-value between those built with reduce LR #0 and their own prebuilt equivalents.

An ANOVA performed on DenseNet-BC tests, however, showed that differences between self-constructed and prebuilt accuracies are in fact statistically significant (if the α value is set to 0.05). The p-value obtained for reduce LR #1 tests was 0.0139, while that for reduce LR #0 tests was 0.02672. Nevertheless, networks generated with reduce LR #0 perform only slightly better than their prebuilt equivalents, while networks generated with reduce LR #1 perform only slightly worse than their own equivalents.

B. Results Obtained with SVHN

The results obtained with SVHN can be found in Table III and their equivalents for prebuilt networks can be found in Table IV. Again, all values were rounded to two decimals except the final number of epochs (rounded to the closest integer).

The validation sets used for SVHN consist of 6,000 random examples from the training set. As with CIFAR-10, the validation set results on both tables concern the validation set created for the last epoch, and the test set results concern the entire actual test set for the SVHN database.

For this dataset, there is a clear difference between the results obtained with DenseNet and those obtained with DenseNet-BC.

In tests with DenseNet, the results in terms of mean accuracy are above 90%, which is quite close both to the state of the art (99% accuracy for the SVHN test set) [22] and to the results obtained from their prebuilt equivalents. This said, the topologies that are produced tend to be very deep, and not as homogenous in their depth as for the tests with CIFAR-10.

The best results with regards to accuracy were those obtained with reduce LR #1. The networks generated by this version of the algorithm were on average the most complex of those generated in the tests, with about 20 layers on average and a standard deviation of 5.50 layers. Nevertheless, they had a mean accuracy of 92.87% on the SVHN test set (vs. 93.02% mean accuracy for prebuilt DenseNet with 20 layers).

TABLE III. SELF-STRUCTURING TESTS USING SVHN

Self-Structuring Tests Using SVHN			Test Results					
			Number of Layers	*End Epoch*	*Validation Set*		*Test Set*	
					Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*
Dense Net	Reduce LR #1	Mean	19.80	409	99.99	0.00	92.87	0.31
		STD	5.50	73	0.02	0.00	0.54	0.02
	Reduce LR #0	Mean	15.40	361	99.96	0.00	92.02	0.35
		STD	5.08	57	0.07	0.00	0.72	0.01
Dense Net-BC	Reduce LR #1	Mean	4.20	236	81.45	0.59	71.81	1.01
		STD	2.68	30	12.56	0.42	10.13	0.40
	Reduce LR #0	Mean	3.00	222	81.76	0.55	70.70	0.99
		STD	0.00	0	3.05	0.08	1.57	0.08

TABLE IV. PREBUILT ARCHITECTURE TESTS CORRESPONDING TO SELF-STRUCTURED ARCHITECTURES, USING SVHN

Prebuilt Architecture Tests Using SVHN			Test Results			
			Validation Set		*Test Set*	
			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*
Dense Net	20 layers (Reduce LR #1)	Mean	100.00	0.00	93.02	0.31
		STD	0.00	0.00	0.17	0.01
	15 layers (Reduce LR #0)	Mean	100.00	0.00	92.59	0.34
		STD	0.00	0.00	0.13	0.01
Dense Net-BC	4 layers (Reduce LR #1)	Mean	92.25	0.23	79.34	0.84
		STD	1.82	0.05	1.77	0.10
	3 layers (Reduce LR #0)	Mean	86.74	0.41	73.73	0.95
		STD	5.11	0.16	3.88	0.19

A mean accuracy of 71.81% on the test set was obtained with reduce LR #1 (vs. 79.34% mean accuracy for prebuilt DenseNet-BC with 4 layers), and one of 70.70% was obtained with reduce LR #0 (vs. 73.73% mean accuracy for DenseNet-BC with 3 layers).

However, the differences that were found between the mean accuracies of self-structured networks and their prebuilt equivalents were not statistically significant. An ANOVA performed on the tests with DenseNet shows a *p*-value of 0.5611 between networks generated with reduce LR #1 and their equivalents, and one of 0.1143 between networks generated with reduce LR #0 and their own equivalents. The results of the ANOVA for tests with DenseNet-BC also obtained *p*-values above 0.05: the *p*-value obtained for reduce LR #1 was 0.1402, and that obtained for reduce LR #0 was 0.1433.

The low accuracy levels for DenseNet-BC are most likely due to the very small number of layers produced, as the only obtained DenseNet-BC with more than three layers (9 layers) had a final accuracy of 87.58% on the SVHN test set. This phenomenon occurs when one of the early layers settles too quickly (its LCS reaches 1 too soon), which seems to be very common for DenseNet-BC trained on SVHN.

C. Results Obtained With Modified Ascension Stage

Normally, there are two ways in which the ascension stage can end: either when one of the layers' LCS reaches 1, or when the network's accuracy on the validation set has not changed much in the latest epochs.

Since the previous experimental results showed that the first of these requirements makes the ascension stage end too soon in some cases (i.e. DenseNet-BC trained on the SVHN dataset), a version of the algorithm without this requirement was tested.

The previous (self-structuring) tests were run again on a version of the algorithm that used a modified ascension stage. The results obtained with CIFAR-10 for this version of the algorithm are found in Table V, and those obtained with SVHN are found in Table VI.

In the case of CIFAR-10, the differences between the accuracies obtained in these tests and those obtained in the original algorithm are not significant, although the *p*-value for DenseNet-BC with reduce LR #1 was of 0.0544. For that case, the mean accuracy on the test set increased slightly (78.29% for the standard algorithm vs. 81.81% for the version with the modified ascension stage).

This increase is probably due to a greater number of layers obtained with the new ascension stage (an average of nearly 12 layers rather than 10). Nevertheless, although there is also an increase in the layer count for DenseNet-BC with reduce LR #0 (11 layers rather than 8), for this combination and all the other ones the differences in accuracy are not statistically significant.

When it comes to SVHN, the results of the ANOVA show that the differences in accuracy are highly significant for all the tests with DenseNet-BC (a *p*-value of 0.0027 for reduce LR #1, and one of $3 * 10^{-9}$ for reduce LR #0). Once again, the most likely explanation for this is a higher number of layers: the new ascension stage does not tend to get stuck at three layers anymore, and therefore is able to reach higher levels of accuracy (91.29% for reduce LR #1, and 92.22% for reduce LR #0). The difference is also almost significant for the tests with DenseNet and reduce LR #1 (the *p*-value was 0.0561).

TABLE V. SELF-STRUCTURING TESTS USING CIFAR-10 (MODIFIED ASCENSION STAGE)

Self-Structuring Tests Using CIFAR-10 (Modified Ascension Stage)			Test Results					
			Number of Layers	*End Epoch*	*Validation Set*		*Test Set*	
					Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*
Dense Net	Reduce LR #1	Mean	7.00	331	76.82	0.75	76.61	0.75
		STD	0.71	86	3.61	0.18	3.68	0.19
	Reduce LR #0	Mean	6.80	316	77.86	0.68	76.60	0.71
		STD	0.45	48	4.01	0.14	4.38	0.15
Dense Net-BC	Reduce LR #1	Mean	12.20	535	82.57	0.58	81.81	0.60
		STD	2.17	195	2.03	0.10	1.76	0.10
	Reduce LR #0	Mean	11.00	531	83.19	0.53	82.26	0.56
		STD	0.71	128	1.96	0.07	1.90	0.07

TABLE VI. SELF-STRUCTURING TESTS USING SVHN (MODIFIED ASCENSION STAGE)

Self-Structuring Tests Using SVHN (Modified Ascension Stage)			Test Results					
			Number of Layers	*End Epoch*	*Validation Set*		*Test Set*	
					Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*
Dense Net	Reduce LR #1	Mean	15.00	347	99.96	0.00	92.11	0.34
		STD	2.92	27	0.04	0.00	0.53	0.02
	Reduce LR #0	Mean	13.80	335	99.95	0.01	91.83	0.35
		STD	2.59	26	0.06	0.00	0.33	0.02
Dense Net-BC	Reduce LR #1	Mean	13.80	334	99.63	0.02	91.29	0.34
		STD	4.44	44	0.30	0.01	1.03	0.04
	Reduce LR #0	Mean	18.00	378	99.90	0.01	92.22	0.31
		STD	3.39	34	0.10	0.00	0.66	0.03

V. CONCLUSIONS AND FUTURE WORK

A new self-structuring algorithm has been developed that is able to select DenseNet and DenseNet-BC structures for a given problem, and build them at the same time as they are being trained. The CNN architectures generated by this algorithm are able to obtain very good accuracy in some of the most popular image classification datasets (CIFAR-10 and SVHN). It is expected that future versions of this algorithm will be able to produce state-of-the-art CNN, capable of competing with handcrafted topologies in terms of their accuracy and simplicity.

After examining the experimental results obtained from this algorithm, one likely route for future work has been identified: adding some form of pruning mechanism after the current two stages (ascension and improvement). In this stage, the connections with lowest CS would be pruned away, thus simplifying the network and perhaps allowing it to reach a higher accuracy.

A version of the algorithm that builds more than one dense block could also be beneficial in terms of accuracy, but this could make networks undesirably deep and complex.

ACKNOWLEDGMENTS

This work is part of a PhD thesis titled “Self-Optimisation of Neural Network Architectures”, carried out at the IRIDIA-CoDE department (Université libre de Bruxelles, ULB).

REFERENCES

[1] S. Waugh, Dynamic Learning Algorithms, Department of Computer Science, University of Tasmania, 1994.

[2] I. Basheer, and M. Hajmeer, “Artificial neural networks: fundamentals, computing, design, and application,” Journal of Microbiological Methods, vol. 43, pp. 3–31, Elsevier, 2000.

[3] S. Duffner, and C. Garcia, “An online backpropagation algorithm with validation error-based adaptive learning rate,” ICANN International Conference of Artificial Neural Networks, Porto, Portugal, 2007.

[4] F. Thabtah, R.M. Mohammad, and L. McCluskey, “A dynamic self-structuring neural network model to combat phishing,” IJCNN International Joint Conference on Neural Networks, pp. 4221–4225, 2016.

[5] R.M. Mohammad, F. Thabtah, and L. McCluskey, “Predicting phishing websites based on self-structuring neural network,” Neural Computing and Applications, vol. 25, issue 2, pp. 443–458, 2013.

[6] M. Islam, A. Sattar, F. Amin, X. Yao, and K. Murase, “A new adaptive merging and growing algorithm for designing artificial neural networks,” IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics. vol. 39, issue 3, pp. 705–722, 2009.

[7] G. Bender, P.J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, “Understanding and simplifying one-shot architecture search,” Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

[8] G. Huang, Z. Liu, L. Van der Maaten, and K.Q. Weinberger, “Densely connected convolutional networks,” arXiv:1608.06993v5, 2016, last updated on 28 January 2018.

[9] F. Thabtah, R.M. Mohammad, and L. McCluskey, “An improved self-structuring neural network, trends and applications in knowledge discovery and data mining,” PAKDD 2016 Workshops, BDM, MLSDA, PACC, WDMBF, Revised Selected Papers, Auckland, New Zealand, pp. 35–47, 2016.

[10] T. Ash, “Dynamic node creation in backpropagation networks,” Connection Science, vol. 1, issue 4, pp. 365–375, 1989.

[11] S.E. Fahlman, and C. Lebiere, “The cascade-correlation learning architecture,” Advances in neural information processing systems, pp. 524–532, 1990.

[12] L. Ma, and K. Khorasani, “A new strategy for adaptively constructing multilayer feedforward neural networks,” Neurocomputing, vol. 31, pp. 361–385, Elsevier, 2003.

[13] T.Y. Kwok, and D.T., Yeung, “Constructive algorithms for structure learning in feedforward neural networks for regression problems,” IEEE Transactions on Neural Networks, vol. 8, issue 3, pp. 630–645, 1997.

[14] S.H. Yang, and Y.P. Chen, “An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications,” Neurocomputing, vol. 86, pp. 140–149, Elsevier, 2012.

[15] T. Salomé, and H. Bersini, “An algorithm for self-structuring neural net classifiers,” IRIDIA, Université Libre de Bruxelles, 1994.

[16] K. He, X. Zhang, S., Ren, and J. Sun., “Deep residual learning for image recognition,” arXiv:1512.03385v1, 2015.

[17] X. Zhang, N. Vishwamitra, H. Hu, and F. Luo., “CrescendoNet: a new deep convolutional neural network with ensemble behavior,” arXiv:1710.11176v2, 2017, last updated on 4 January 2018.

[18] I. Khlestov, “DenseNet with TensorFlow,” GitHub repository. https://github.com/ikhlestov/vision_networks, 2017, last updated on 31 July 2018.

[19] A. García Díaz, “Study of deep learning algorithms: incremental solutions,” Master thesis, Université Libre de Bruxelles, unpublished. https://github.com/AntonioGarciaDiaz/MasterThesis-IncrementalDeepLearning, 2018.

[20] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Master thesis, Department of Computer Science, University of Toronto.

[21] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A.Y. Ng, “Reading digits in natural images with unsupervised feature learning,” NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

[22] https://paperswithcode.com/task/image-classification.