Evolving Robust Neural Architectures to Defend from Adversarial Attacks

Shashank Kotyan and **Danilo Vasconcellos Vargas**
Department of Informatics, Kyushu University, Japan
shashankkotyan@gmail.com and vargas@inf.kyushu-u.ac.jp

Abstract

Neural networks are prone to misclassify slightly modified input images. Recently, many defences have been proposed, but none have improved the robustness of neural networks consistently. Here, we propose to use adversarial attacks as a function evaluation to search for neural architectures that can resist such attacks automatically. Experiments on neural architecture search algorithms from the literature show that although accurate, they are not able to find robust architectures. A significant reason for this lies in their limited search space. By creating a novel neural architecture search with options for dense layers to connect with convolution layers and vice-versa as well as the addition of concatenation layers in the search, we were able to evolve an architecture that is inherently accurate on adversarial samples. Interestingly, this inherent robustness of the evolved architecture rivals state-of-the-art defences such as adversarial training while being trained only on the non-adversarial samples. Moreover, the evolved architecture makes use of some peculiar traits which might be useful for developing even more robust ones. Thus, the results here confirm that more robust architectures exist as well as opens up a new realm of feasibilities for the development and exploration of neural networks.

1 Introduction

Neural Architecture Search (NAS) and adversarial samples have rarely appeared together. Regarding adversarial samples, they were discovered in 2013 when neural networks were shown to behave strangely for nearly the same images [Szegedy, 2014]. Afterwards, a series of vulnerabilities were found in [Moosavi-Dezfooli *et al.*, 2017; Su *et al.*, 2019]. Such adversarial attacks can also be easily applied to real-world scenarios which confer a big problem for current deep neural networks' applications. Currently, there is not any known learning algorithm or procedure that can defend against adversarial attacks *consistently*.

Regarding NAS, the automatic design of architectures has been of broad interest for many years. The aim is to develop methods that do not need specialists in order to be applied to a different application. This would confer not only generality but also easy of use. Most of the algorithms for NAS are either based on reinforcement learning [Pham *et al.*, 2018; Zoph *et al.*, 2018] or evolutionary computation [Real *et al.*, 2017; Miikkulainen *et al.*, 2019]. On the one hand, in reinforcement learning approaches, architectures are created from a sequence of actions which are afterwards rewarded proportionally to the crafted architecture's accuracy. On the other hand, in evolutionary computation based methods, small changes in the architecture (mutations) and recombinations (crossover) are used to create new architectures. All architectures evolved are evaluated based on their accuracy. Some of the best architectures based on this accuracy are chosen to continue to the next generation.

Here we propose the use of NAS to tackle the robustness issues exposed by adversarial samples. In other words, architecture search will be employed not only to find accurate neural networks but also robust ones. This is based on the principle that robustness of neural networks can be assessed by using accuracy on adversarial samples as an evaluation function. We hypothesise that if there is a solution in a given architecture search space, the search algorithm would be able to find it. This is not only a blind search for a cure. The best architectures found should also hint which structures and procedures provide robustness for neural networks. Therefore, it would be possible to use the results of the search to understand further how to improve the representation of models as well as design yet more robust ones.

2 Adversarial Machine Learning

Adversarial machine learning is a constrained optimisation problem. Let $f(x) \in [\![1..N]\!]$ be the output of a machine learning algorithm in multi-label classification setting. Here, $x \in \mathbb{R}^k$ is the input of the algorithm for the input of size k and N is the number of classes in which x can be classified. In Image Classification problem $k = m \times n \times 3$ where $m \times n$ is the the size of the image. Adversarial samples x' can be thus defined as follows:

$$x' = x + \epsilon_x \quad \text{such that} \quad f(x') \neq f(x) \tag{1}$$

in which $\epsilon_x \in \mathbb{R}^k$ is a small perturbation added to the input. Therefore, adversarial machine learning can be defined as an optimization problem[1]:

$$\underset{\epsilon_x}{\text{minimize}} \quad g(x+\epsilon_x)_c \quad \text{subject to} \quad \|\epsilon_x\| \leq th \tag{2}$$

where th is a pre-defined threshold value and $g()_c$ is the soft-label or confidence for the correct class c such that $f(x) = \text{argmax } g(x)$

Moreover, attacks can be divided according to the function optimised. In this way, there are L_0 (limited number of pixels attacked), L_1, L_2 and L_∞ (limited amount of variation in each pixel) types of attacks. There are many types of attacks as well as their improvements. Universal perturbation types of attacks were shown possible in which a single perturbation added to most of the samples is capable of fooling a neural network in most of the cases [Moosavi-Dezfooli *et al.*, 2017]. Moreover, extreme attacks such as only modifying one pixel ($L_0 = 1$) called one-pixel attack is also shown to be surprisingly effective [Su *et al.*, 2019; Vargas and Kotyan, 2019]. Most of these attacks canbe easily transferred to real scenarios by using printed out versions of them [Kurakin *et al.*, 2016]. Moreover, carefully crafted glasses [Sharif *et al.*, 2016] or even general 3D adversarial objects are also capable of causing misclassification [Athalye and Sutskever, 2018]. Regarding understanding the phenomenon, it is argued in [Goodfellow *et al.*, 2014] that neural networks' linearity is one of the main reasons. Another recent investigation proposes the conflicting saliency added by adversarial samples as the reason for misclassification [Vargas and Su, 2019].

Many defensive systems were proposed to mitigate some of the problems. However, current solutions are still far from solving the problems. Defensive distillation [Papernot *et al.*, 2016] uses a smaller neural network to learn the content from the original one; however, it was shown not to be robust enough [Carlini and Wagner, 2017]. The addition of adversarial samples to the training dataset, called adversarial training, was also proposed [Goodfellow *et al.*, 2014; Huang *et al.*, 2015; Madry *et al.*, 2018]. However, adversarial training has a strong bias in the type of adversarial samples used and is still vulnerable to attacks Many recent variations of defences were proposed which are carefully analysed, and many of their shortcomings explained in [Athalye *et al.*, 2018; Uesato *et al.*, 2018]. In this article, different from previous approaches, we aim to tackle the robustness problems of neural networks by automatically searching for inherent robust architectures.

3 Neural Architecture Search

There are three components to a neural architecture search: search space, search strategy and performance estimation. A search space substantially limits the representation of the architecture in a given space. A search strategy must be employed to search for architectures in a defined search space. Some widely used search strategies for NAS are: Random Search, Bayesian Optimization, Evolutionary Methods, Reinforcement Learning, and Gradient Based Methods. Finally, a performance estimation (usually error rate) is required to evaluate the explored architectures.

Currently, most of the current NAS suffer from high computational cost while searching in a relatively small search space [Lee *et al.*, 2018; Lima and Pozo, 2019].It is already shown in [Yu and Kim, 2018] that, if there is a possibility of fitness approximation at small search spaces, we could evolve algorithms in an ample search space. Moreover, many architecture searches focus primarily on the hyper-parameter search while using architecture search spaces around previously hand-crafted architecture [Lee *et al.*, 2018; He *et al.*, 2019] such as DenseNet which are proved to be vulnerable to adversarial attacks [Vargas and Kotyan, 2019]. Therefore, for finding robust architectures, it is crucial to expand the search space beyond the current NAS.

SMASH [Brock *et al.*, 2017] uses a neural network to generate the weights of the primary model. The main strength of this approach lies in preventing high computational cost, which is incurred in other searches. However, this comes at the cost of not being able of tweaking hyperparameters which affect weights like initialisers and regularisers. Deep Architect [Negrinho and Gordon, 2017] follows a hierarchical approach using various search algorithms such as Monte Carlo Tree Search (MCTS) and Sequential Model based Global Optimization (SMBO).

4 Searching for Robust Architectures

A robust evaluation (defined in Section 4.1) and search algorithm must be defined to search for robust architectures. The search algorithm may be a NAS provided that some modifications are made (Section 4.2). However, to allow for a more extensive search space, which is better suited to the problem, we also propose the Robust Architecture Search (Section 5).

4.1 Robustness Evaluation

Adversarial accuracy may seem like a natural evaluation function for assessing neural networks' robustness. However, there are many types of perturbations possible; each will result in a different type of robustness assessment and evolution. For example, let us suppose an evaluation with $th = 5$ is chosen, robust networks against $th = 5$ might be developed. At the same time, nothing can be said for other th and attack types (different L). Therefore, th plays a role but the different types of L_0, L_1, L_2 and L_∞ completely change the type of robustness, such as wide perturbations (L_∞), punctual perturbations (L_0) and a mix of both (L_1 and L_2). To avoid creating neural networks that are only robust against one type of robustness and at the same time to allow robustness to slowly build-up from any partial robustness, a set of adversarial attacks for varying th and L are necessary.

To evaluate the robustness of architectures in varying th and L while at the same time keeping computational cost low, we use here a transferable type of attack. In other words, adversarial samples previously found by attacking other methods are stored and used as possible adversarial samples to the current model under evaluation. This solves the problem that

[1]Here the definition will only concern untargeted attacks but a similar optimization problem can be defined for targeted attacks

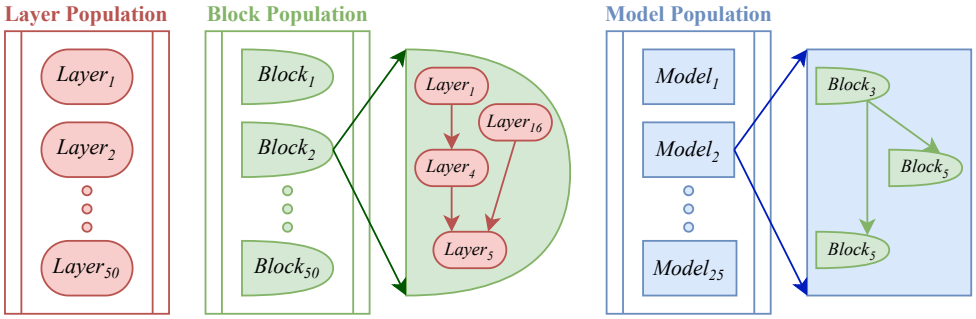

Figure 1: Illustration of the proposed RAS structure with three subpopulations.

Model	Attack Optimiser	L_0 Attack				L_∞ Attack				*Total*
		$th = 1$	$th = 3$	$th = 5$	$th = 10$	$th = 1$	$th = 3$	$th = 5$	$th = 10$	
CapsNet	DE	18	46	45	47	05	09	12	24	*206*
	CMA-ES	14	34	45	62	09	38	74	98	*374*
ResNet	DE	23	66	75	77	06	22	46	78	*393*
	CMA-ES	11	49	63	77	28	72	75	83	*458*
AT	DE	23	59	63	66	00	02	03	06	*222*
	CMA-ES	20	50	70	82	03	12	25	57	*319*
FS	DE	21	73	78	78	04	21	45	78	*398*
	CMA-ES	17	49	69	78	26	63	66	74	*442*
	Total	*147*	*426*	*508*	*567*	*81*	*239*	*346*	*498*	*2812*

Table 1: The number of samples used from each type of black-box attack to compose the 2812 adversarial samples. Based on the principle of the transferability of adversarial samples, these adversarial samples are used as a fast attack for the robustness evaluation of architectures. Details of the attacks as well as the motivation for using a model-agnostic (black-box) dual quality (L_0 and L_∞) assessment are explained in detail at [Vargas and Kotyan, 2019].

most of the attacks are usually slow to be put inside a loop which can make the search for architectures too expensive.

Table 1 shows a summary of the number of images used from each type of attack, totalling 2812 adversarial samples. Samples were generated using the model agnostic dual quality assessment [Vargas and Kotyan, 2019]. Specifically, we use the adversarial samples from two types of attacks (L_0 and L_∞ attacks) with two optimization algorithms (Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Hansen *et al.*, 2003] and Differential Evolution (DE) [Storn and Price, 1997]). We use CIFAR-10 dataset [Krizhevsky *et al.*, 2009] to generate the adversarial samples. We attacked traditional architectures such as ResNet [He *et al.*, 2016] and CapsNet [Sabour *et al.*, 2017]. We also attacked some state-of-art-defences such as Adversarial Training (AT) [Madry *et al.*, 2018] and Feature Squeezing (FS) [Xu *et al.*, 2017] defending ResNet. The evaluation procedure consists of calculating the amount of successful adversarial samples divided by the total of possible adversarial samples. This also avoids problems with different amount of perturbation necessary for attacks to succeed, which could cause incomparable results.

4.2 Robust Search Conversion of existing NAS

By changing the fitness function (in the case of evolutionary computation based NAS) or the reward function (in the case of reinforcement learning based NAS), it is possible to create robust search versions of NAS algorithms. In other words, it is possible to convert the search for accuracy into the search for robustness and accuracy. Here we use SMASH and DeepArchitect for the tests. The reason for the choice lies in the difference between the methods and availability of the code. Both methods have their evaluation function modified to contain not only accuracy but also robustness (Section 4.1).

5 Robust Architecture Search (RAS)

Here, we propose an evolutionary algorithm to search for robust architectures called Robust Architecture Search (RAS)[2]. It makes sense to focus here on search spaces that allow for unusual layer types and their combinations to happen, which is more vast than the current traditional search spaces. The motivation to consider this vast search space is that some of the most robust architectures might contain these unusual combinations which are not yet found or deeply explored.

RAS Overview RAS works by creating three initial populations (layer, block and model populations). Every generation, the model population have each of its members modified five times by mutations. The modified members are added to the population as new members. Here we propose a utility evaluation in which layer and block populations are evaluated by the number of models (architectures) using them. Models are evaluated by their accuracy and attack resilience (accuracy on adversarial samples). All blocks and layers which are

[2]Code is available at http://bit.ly/RobustArchitectureSearch

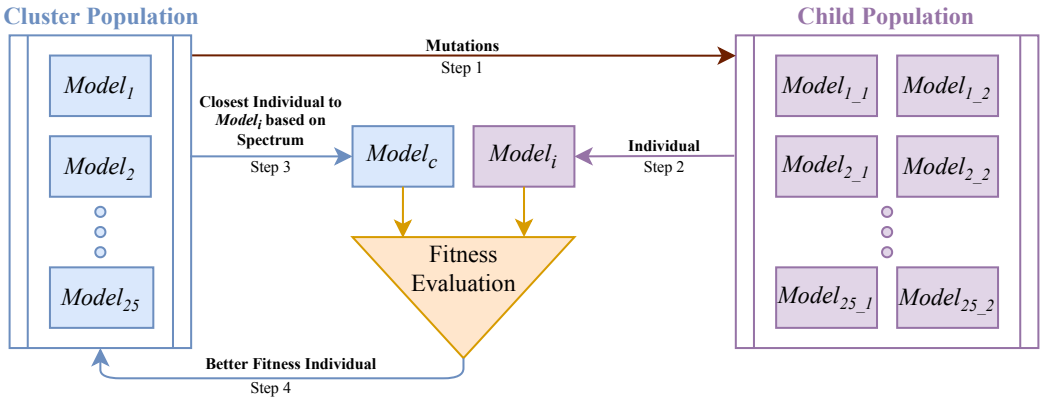

Figure 2: Illustration of the proposed Evolutionary Strategy. In this strategy, Step 2, 3, and 4 are repeated for all the individuals in the child population. Step 1 is repeated when all the individuals in the child population have been evaluated against a cluster individual.

not used by any of the current members of the model population are removed at the end step of each generation. Moreover, architectures compete with similar ones in their subpopulation, such that only the fittest of each subpopulation survives.

5.1 Description of Population

For such a vast search space to be more efficiently searched, we propose to use three subpopulations, allowing for the reuse of blocks and layers. Specifically, the layers consist of: *Layer Population:* Raw layers (convolutional and fully connected) which make up the blocks. *Block Population:* Blocks which are a combination of layers. *Model Population:* A population of architectures which consists of interconnected blocks. Figure 1 illustrates the architecture.

The initial population consists of 25 random architectures which contain $U(2, 5)$ blocks made up of $U(2, 5)$ layers, in which $U(a, b)$ is a uniform random distribution with minimum a and maximum b values. The possible available parameters for the layers are as follows: for convolutional layers, filter size might be 8, 16, 32 or 64, stride size may be 1 or 2 and kernel size is either 1, 3 or 5; for fully connected layers, the unit size may assume the values of 64, 128, 256 or 512. All the layers use Rectified Linear Unit (ReLU) as an activation function and are followed by a batch-normalisation layer.

5.2 Mutation Operators

Regarding the mutation operators used to evolve the architecture, they can be divided into layer, block and model mutations which can only be applied to the respective layer, block and model populations' individuals. The following paragraphs define the possible mutations.

Layer Mutation Layer mutations are of the following types: *a) Change Kernel:* Changes the kernel size of the convolution layer, *b) Change Filter:* Changes the filter size of the convolution layer, *c) Change Units:* Changes the unit size of the fully connected layer, *d) Swap Layer:* Chosen layer is swapped with a random layer from the layer population.

Block Mutation Block mutation change a single block in the block population. The possibilities are: *e) Add Layer:* A random layer is added to a chosen random block, *f) Remove Layer:* A random layer is removed from a chosen random block, *g) Add Layer Connection:* A random connection between two layers from the chosen random block is added, *h) Remove Layer Connection:* A random connection between the two layers from the chosen random block is removed, *i) Swap Block:* Chosen block is swapped with a random block from the population.

Model Mutation Model mutation modify a given architecture. The possible model mutations are: *j) Add Block:* A random block is added to the model, *k) Remove Block:* A random block is removed from the model, *l) Add Block connection:* A random connection between the two blocks is added, *m) Remove Block connection:* A random connection between the two blocks is removed.

All mutations add a new member to the population instead of substituting the previous one. In this manner, if nothing is done, the population of layers and blocks may explode, increasing the number of lesser quality layers and blocks. This would cause the probability of choosing functional layers and blocks to decrease. To avoid this, when the layer or block population exceeds 100 individuals, the only layer/block mutation available is swap layers/blocks.

5.3 Objective (Fitness) Function

Fitness of an individual of the model population is measured using the final validation accuracy of the model after training for a maximum of 200 epochs with early stopping if accuracy or validation accuracy do not change more than 0.001 in the span of 15 epochs. Regarding the fitness calculation, the fitness is calculated as the accuracy of the model plus the robustness of the model (**Fitness** = **Accuracy** + **Robustness**).

The **Accuracy** of the architecture is calculated after the model is trained for 50 epochs over the whole set of samples (50000 samples) of the CIFAR-10's training dataset for every 10 generation(s) or over 1000 random samples of the CIFAR-10 training dataset for all other generations. This allows an efficient evolution to happen in which blocks and lay-

ers evolve at a faster rate without interfering with the architecture's accuracy. Using entire dataset subjects to evolving the architecture to have better accuracy and using a subset of the dataset evolves the layers and blocks of the architecture at a faster rate. The **Robustness** of the architecture is calculated using accuracy on adversarial samples as described in Section 4.1.

5.4 Spectrum-based Niching Scheme

To keep a high amount of diversity while searching in a vast search space by using a novel algorithm described below also shown in Figure 2. This niching scheme uses the idea of Spectrum-based niching from [Vargas and Murata, 2017] but explores a different approach to it. First, all the initial population is converted into a cluster population such that each individual in the initial population is a cluster representative. Then we create two child individuals for each cluster representative by randomly applying five mutation operators on cluster representative. We then find the closest cluster representative to the child individual using spectrum described below. If the fitness of the child individual is better than the closest cluster representative than the child individual becomes the new cluster representative, and the old cluster representative is removed from the population and the generation. The process is completed for all the individuals in a cluster population. We are hence evolving a generation of the evolution.

Here, we use the spectrum as a histogram containing the features: Number of Blocks, Number of Total Layers, Number of Block Connections, Number of Total Layer Connections, Number of Dense Layers, Number of Convolution Layers, Number of Dense to Dense Connections, Number of Dense to Convolution Connections, Number of Convolution to Dense Connections, and Number of Convolution to Convolution Connections. By using this Spectrum-based niching scheme, we aim to achieve an open-ended evolution, preventing the evolution from converging to a single robust architecture. Preserving diversity in the population ensures that the exploration rate remains relatively high, allowing us to find different architectures even after many evolution steps. For the vast search space of architectures, this property is especially important, allowing the algorithm to traverse the vast search space efficiently.

6 Experiments on RAS and Converted NAS

Architecture Search	Testing ER	ER on Adversarial Samples
DeepArchitect*	25%	75%
SMASH*	23%	82%
Ours	**18%**	**42%**

Table 2: Error Rate (ER) on both the testing dataset and adversarial samples when the evaluation function has both accuracies on the testing data and accuracy on the adversarial samples.
*Both DeepArchitect and SMASH had their evaluation function modified to be the sum of accuracy on the testing and adversarial samples.

Here, experiments are conducted on both the proposed RAS and converted versions of DeepArchitect and SMASH. The objective is to achieve the highest robustness possible using different types of architecture search algorithms and compare their result and effectiveness. Initially, DeepArchitect and Smash found architectures which had an error rate of 11% and 4% respectively when the fitness is only based on the neural network's testing accuracy. However, when the accuracy on adversarial samples is included in the evaluation function, the final error rate increases to 25% and 23% respectively (Table 2). This may also indicate that poisoning the dataset might cause a substantial decrease in accuracy for the architectures found by SMASH and DeepArchitect. In the case of RAS, even with a more extensive search space, an error rate of 18% is achieved.

Regarding the robustness of the architectures found, Table 2 shows that the final architecture found by DeepArchitect and SMASH were very susceptible to attacks, with error rate on adversarial samples of 75% and 82% respectively. Despite the inclusion of the R (measured accuracy on adversarial samples) on the evaluation function, the architectures were still unable to find a robust architecture. This might be a consequence of the relatively small search space used and more focused initialisation procedures. Moreover, the proposed method (RAS) finds an architecture which has an error rate of only 42% on adversarial samples. *Note, however, that in the case of the evolved architecture, this is an inherent property of the architecture found*. The architecture is inherently robust without any kind of specialised training or defence such as adversarial training (i.e., the architecture was only trained on the training dataset). The addition of defences should increase its robustness further.

7 Analyzing RAS

In this section, we will evaluate the proposed architecture regarding its evolution quality and how subpopulations behave throughout the process. Figure 3 shows how the mean accuracy of the architectures evolved increases over time. The pattern of behaviour is typical of evolutionary algorithms, showing that evolution is happening as expected.

In Figure 4, the overall characteristics of the evolved architectures throughout the generations are shown. The average number of blocks and the connections between them increase over the generations. However, the average number of layers never reaches the same complexity as the initial models. The number of layers decreases steeply initially while slowly increasing afterwards. Therefore, the overall behaviour is that blocks become smaller and numerous. A consequence of this is that the number of connections becomes proportional to the number of connections between blocks and therefore exhibit similar behaviour. The average number of layers per block and the average number of connections shows little change, varying only 0.5 and 0.16 respectively.

Notice that the average number of layers increases but the average number of layers per block continues to decrease albeit slowly. Consequently, blocks tend to degenerate into a few layers, resulting in around three layers per block from the first average number of 3.2 layers per block. Lastly, the

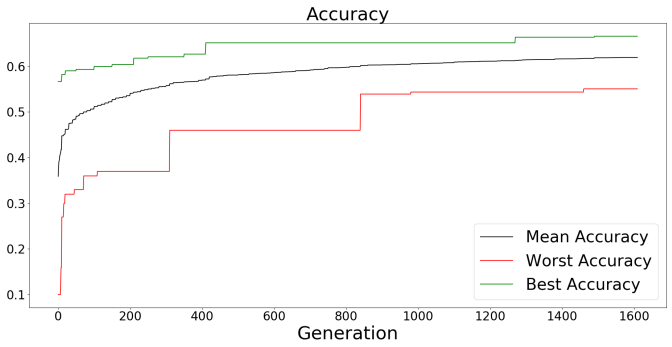

Figure 3: Accuracy improvement over the generations.

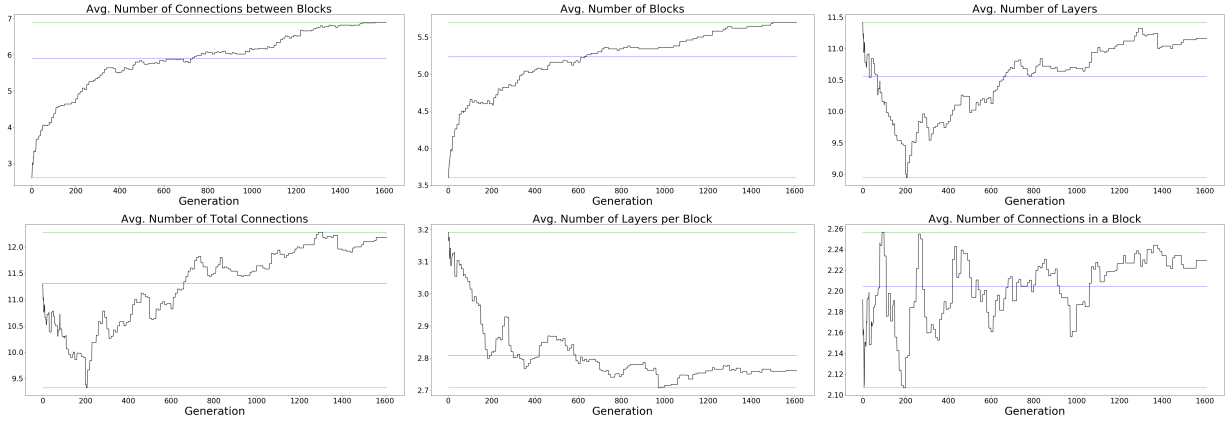

Figure 4: The overall distribution of the architectures found in each generation. The connections from the input layer and the softmax layer are always present and, therefore, they are omitted in the calculation.

average number of connections in a block is kept more or less the same, with the mean varying throughout only from 2.1 to 2.26. The behaviour described above might suggest that it is hard to create big reusable blocks. This seems to be supported by both the decrease of complexity observed as well as the increase in the number of blocks.

8 Analyzing the Final Architecture: Searching for the Key to Inherent Robustness

RAS found an architecture that possesses inherent robustness capable of rivalling current defences. To investigate the reason behind this robustness, we can take a more in-depth look at the architecture found. Figure 5 show(s) some peculiarities from the evolved architecture: multiple bottlenecks, projections into high-dimensional space and paths with different constraints.

Multiple Bottlenecks and Projections into High-Dimensional Space The first peculiarity is the use of Dense layers in-between Convolutional ones. This might seem like a bottleneck similar to the ones used in variational autoencoders. However, it is the opposite of a bottleneck (Figure 5); it is a projection in high-dimensional space. The evolved architecture uses mostly a low number of filters while, in some parts of it, high-dimensional projections exist. In the whole architecture, four Dense layers in-between Convolutional ones were used, and all of the projects into higher dimensional space. This follows directly from Cover's Theorem which states that projecting into high dimensional space makes a training set linearly separable [Cover, 1965].

Paths with Different Constraints The second peculiarity is the use of multiple paths with the different number of filters and output sizes after high-dimensional projections. Notice how the number of filters differs in each of the Convolutional layers in these paths. This means there are different constraints over the learning in each of these paths, which should foster different types of features. Therefore, this is a multi-bottleneck structure forcing the learning of different sets of features which are now easily constructed from the previous high-dimensional projection.

9 Conclusions

Automatic search for robust architectures is proposed as a paradigm for developing and researching robust models. This paradigm is based on using adversarial attacks together with error rate as evaluation functions in NAS. Experiments on using this paradigm with some of the current NAS had poor results. This was justified by the small search space used by current methods. Here, we propose the RAS method,

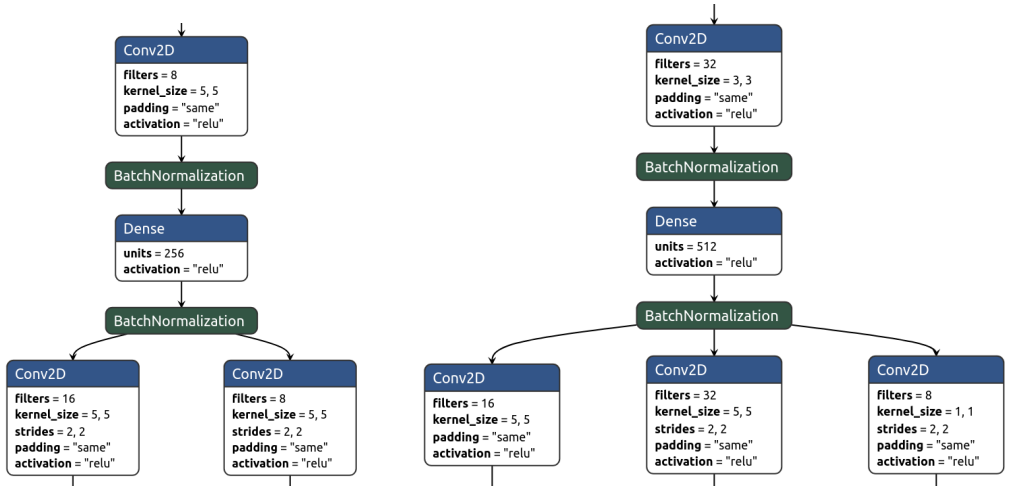

Figure 5: Two fragments of the evolved architecture which has peculiar traits.

which has a broader search space, including concatenation, connections between dense to convolutional layer and vice-versa. Results with RAS showed that inherently robust architectures do indeed exist. In fact, the evolved architecture achieved robust results comparable with state-of-the-art defences while not having any specialised training or defence. In other words, the evolved architecture is *inherently robust*. Such inherent robustness could increase if adversarial training, or other types of defence, or a combination of them are employed together with it.

Moreover, investigating the reasons behind such robustness have shown that some peculiar traits are present. The evolved architecture has overall a low number of filters and many bottlenecks. Multiple projections into high-dimensional space are also present to possibly facilitate the separation of features (Cover's Theorem). It also uses multiple paths with different constraints after the high-dimensional projection, which should, consequently, cause a diverse set of features to be learned by the network. Thus, in the search space of neural networks, more robust architectures do exist, and more research is required to find and fully document them as well as their features.

Acknowledgments

This work was supported by JST, ACT-I Grant Number JP-50243 and JSPS KAKENHI Grant Number JP20241216. Additionally, we would like to thank Prof. Junichi Murata for the kind support without which it would not be possible to conduct this research.

References

[Athalye and Sutskever, 2018] Anish Athalye and Ilya Sutskever. Synthesizing robust adversarial examples. In *Icml*, 2018.

[Athalye *et al.*, 2018] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Icml*, 2018.

[Brock *et al.*, 2017] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[Carlini and Wagner, 2017] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. Ieee, 2017.

[Cover, 1965] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3):326–334, 1965.

[Goodfellow *et al.*, 2014] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[Hansen *et al.*, 2003] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[He *et al.*, 2019] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *arXiv preprint arXiv:1908.00709*, 2019.

[Huang *et al.*, 2015] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *arXiv preprint arXiv:1511.03034*, 2015.

[Krizhevsky *et al.*, 2009] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009.

[Kurakin *et al.*, 2016] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[Lee *et al.*, 2018] Hyeon-Chang Lee, Dong-Pil Yu, and Yong-Hyuk Kim. On the hardness of parameter optimization of convolution neural networks using genetic algorithm and machine learning. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 51–52, 2018.

[Lima and Pozo, 2019] Ricardo HR Lima and Aurora TR Pozo. Evolving convolutional neural networks through grammatical evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 179–180, 2019.

[Madry *et al.*, 2018] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Iclr*, 2018.

[Miikkulainen *et al.*, 2019] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[Moosavi-Dezfooli *et al.*, 2017] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94. Ieee, 2017.

[Negrinho and Gordon, 2017] Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

[Papernot *et al.*, 2016] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. Ieee, 2016.

[Pham *et al.*, 2018] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pages 4092–4101, 2018.

[Real *et al.*, 2017] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017.

[Sabour *et al.*, 2017] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[Sharif *et al.*, 2016] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540. Acm, 2016.

[Storn and Price, 1997] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[Su *et al.*, 2019] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[Szegedy, 2014] Christian et al. Szegedy. Intriguing properties of neural networks. In *In ICLR*. Citeseer, 2014.

[Uesato *et al.*, 2018] Jonathan Uesato, Brendan O'Donoghue, Pushmeet Kohli, and Aaron Oord. Adversarial risk and the dangers of evaluating against weak attacks. In *International Conference on Machine Learning*, pages 5032–5041, 2018.

[Vargas and Kotyan, 2019] Danilo Vasconcellos Vargas and Shashank Kotyan. Robustness assessment for adversarial machine learning: Problems, solutions and a survey of current neural networks and defenses. *arXiv preprint arXiv:1906.06026*, 2019.

[Vargas and Murata, 2017] Danilo Vasconcellos Vargas and Junichi Murata. Spectrum-diverse neuroevolution with unified neural models. *IEEE transactions on neural networks and learning systems*, 28(8):1759–1773, 2017.

[Vargas and Su, 2019] Danilo Vasconcellos Vargas and Jiawei Su. Understanding the one-pixel attack: Propagation maps and locality analysis. *arXiv preprint arXiv:1902.02947*, 2019.

[Xu *et al.*, 2017] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[Yu and Kim, 2018] Dong-Pil Yu and Yong-Hyuk Kim. Is it worth to approximate fitness by machine learning? investigation on the extensibility according to problem size. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 77–78, 2018.

[Zoph *et al.*, 2018] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.