EComp: Evolutionary Compression of Neural Networks Using a Novel Similarity Objective

Stijn van den Beemt (s4331354)
Pascal Schröder (s1062138)
Thijme de Valk (s4798902)

1 Introduction

Deciding the architecture of neural networks used in optimisation problems is a difficult task, often done iteratively and intuitively guided by human researchers. Neural architecture search (NAS) aims to automate this task, but is expensive in both time and computational resources. To aid this, several NAS methods have been developed using evolutionary strategies (ENAS) to make the search procedure more informed and thus efficient. However, these techniques are generally still computationally expensive, especially when they rely on gradient descent based retraining of candidate solutions [4, 5].

One interesting example of ENAS methods is network compression. The lottery ticket hypothesis states that neural networks contain subnetworks that are particularly effective for solving the relevant problem, even when they are taken from the network and used in isolation. These subnetworks, that emerge after training with random initialisation, are generally much smaller than the original network but still perform well [2]. Because neural networks are known for their complexity and redundancy, compressing network architectures to make their training and (in the case of large networks particularly) inference more efficient is a promising approach. Several works have investigated this problem [3, 10, 11]. While they show promising results, most of these methods only rely on the error rate of the compressed network to evaluate how well the compression worked. This can be problematic, as optimising for accuracy only may cause implicit retraining of the networks during optimisation. Therefore, severe limitations to the possible recombination and mutation schemes are required to ensure that the found network is really a compressed version of the original network, and not a fully retrained network of smaller size.

In the study of information theory, several measures exist to quantify the amount of useful information between two signals, relative to redundant or less important information. Such measures can also be used to evaluate the goodness of a compressed signal, which aims to retain a maximum of useful information, while disregarding the rest. We hypothesise that such measures can analogously be utilised in the domain of neural network compression, by quantifying how much information a particular hidden layer of a compressed network contains relative to that of the original network. To this end, we implement a custom similarity loss objective based on minimising the Kullback-Leibler divergence between the embedding spaces spanned by the hidden layers of origin and compressed network.

As a research question, we thus ask to what extend the addition of our custom similarity loss to the fitness function of a evolutionary compression algorithm provides effective compression. With effective compression, we mean that the resulting compressed network is truly a compressed version of the original, and not a completely retrained smaller network. In order to provide an answer to this question, we investigate how this similarity loss interacts with the other already existing measures for accuracy and compression. The effects of these interactions on convergence and consistency of results over different runs are analysed, and related to the lottery ticket hypothesis.

We hypothesise that our similarity objective can act as a search heuristic, directing ENAS in the direction of networks that use a similar embedding space. Therefore, we expect the output network to be closely related to the parent network, and effective at solving the original problem. Moreover, we expect that the heuristic effect speeds up convergence to a well performing population consistently over different runs.

2 Aim

Our aim is to provide a proof-of-concept: An ENAS implementation that compresses a neural network, using our proposed custom similarity loss objective as part of the fitness evaluation function. By experimentation, we seek to show that including this similarity measure in the fitness function ensures that the resulting compressed network uses similar hidden layer representations as the parent network. Additionally, we aim to illustrate the potential benefits of our similarity loss for fast and consistent convergence of the evolutionary search process.

3 Methods

We now present EComp (Evolutionary Compressor) for neural network and will explain the experiments we conducted to test its behaviour and performance under different conditions. The full implementation of the EComp algorithm as well as the framework for the experiments can be found on GitHub[1].

3.1 EComp Algorithm

On a superficial level, EComp follows a classic evolutionary strategy that takes a trained balanced (i.e. with all hidden layers the same size) deep feed forward neural network as input and compresses it to a network of smaller hidden layer size, without requiring explicit (re)training in intermediate steps, which is laid out in Algorithm 1. We implemented all networks as well as the parent network's initial training using the PyTorch Deep Learning library [6].

3.1.1 Initialiser

In order to create an initial population from the parent network, the initialiser creates a given number of compressed networks. We take evenly spaced compression rates from an interval between a specified minimal and maximal number of hidden layers, relative to the original parent network's size. Initially, we planned on using truncated singular value decomposition (t-SVD) to reduce the dimensionality of the hidden layer in an computationally expensive but exact way. However, t-SVD is not a suitable algorithm for compression of multi-layer deep networks, as it relies on retraining of deeper layers, for each compressed layer [3]. Using t-SVD would thus restrict the type of networks for which our approach is suited. Therefore, we decided to work with a less elegant but more practical pruning strategy based on L1 loss [1]. This form of pruning is based on a forward pass through the network, and the deletion of less active neurons from the sparse layer representation.

3.1.2 Fitness Calculation

As one of the most critical components of EComp, the fitness function is a linear combination of three loss measures, such that $fitness = w_a * a - w_s * s + w_c * (1 - c)$, where w_m means weight for measure m and a, s, c are scalar values representing the outcomes of the accuracy, compression and similarity loss measures for a specific network respectively.

Accuracy Measure In order to quantify the predictive performance of the network, we track the accuracy (number of true positives divided by total predictions) over a batch of the training data. A new batch is fed at every generation of the EComp algorithm, to avoid overfitting to a certain reused batch. Thus, in each generation, the accuracy of all networks in the population is determined on the same batch and reported back to use in the networks fitness function.

Compression Measure As a compression measure we take the relative size of the child network compared to the parent network, i.e. the number of weights in the first hidden layer of the network divided by those of the original parent network. Only using this first layer is possible because of the aforementioned constraint that both our parent network and all our children networks have to be balanced (all hidden layers have the same size), which is further explained in the description of the recombiner in section 3.1.4.

Similarity Measure The problem with designing an objective function to quantify the relative information between the embedding spaces of a compressed network and its origin network lies in the different dimensionality of both spaces. Since the goal of neural network compression is a reduction in layer size, the embedding space of a compressed network will always be of lower dimensionality. Applying measures of information entropy, like information gain, mutual information, or Kullback-Leibler divergence, to the outputs of the networks' hidden layers thus does not work, as such measures require probability distribution of equal dimensionality. To circumvent this problem, we took inspiration from the t-distributed Stochastic Neighbourhood Embeddings (t-SNE) algorithm [8].

[1] https://github.com/verrannt/enas-compression/

Algorithm 1: The EComp Algorithm

Let p_{mutate} the mutation rate, and D the training data
Input: A feed-forward neural network with balanced (i.e. equal) hidden layer sizes, trained on D
Create initial population P_0 using network pruning ;
For each compressed network $c_n \in P_0$, compute initial fitnesses $f_n(b_0, c_n) \in F_0$, with b_0 denoting the first batch in D ;
for *each epoch* **do**
  for *each batch* b_i *in* D **do**
    Select candidates from population P_i based on fitnesses F_i ;
    Recombine candidates into a new population P_{i+1} ;
    For each $c_n \in P_{i+1}$, mutate with chance p_{mutate} ;
    For each $c_n \in P_{i+1}$, compute fitness $f_n(b_i, c_n) \in F_{i+1}$;
  end
end

t-SNE is an algorithm for dimensionality reduction, which employs the Kullback-Leibler divergence to maximise the informational similarity between the original, high-dimensional data space and the low-dimensional embedding. In short, it achieves this by creating two probability distribution over pairwise distances, one for the original space, as well as one for a random low-dimensional space. The Kullback-Leibler divergence is then used to change the low-dimensional representations of each data point, such that their distribution over pairwise distances approximates that of the high-dimensional space. This circumvents the problem of the different dimensionality of both spaces, since the amount of pairwise distances depends only on the amount of data points, not on the dimensionality of the space in which they are represented.

We copy this approach, in that we define the parent network's embedding space to be what is the original data space for t-SNE, and the compressed network's embedding space to be the lower-dimensional space. We then define the two probability distributions over pairwise distances for each embedding space. While t-SNE uses the Kullback-Leibler divergence as a loss function to iteratively optimise the lower-dimensional distribution, we only use it as a single-step evaluation, since we do not intend to change the compressed network's weights, but desire to only evaluate the networks.

3.1.3 Selector

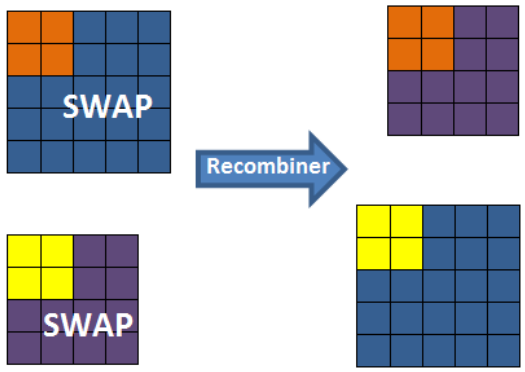
To create the mating pools, the selection method we settled with was a fairly simple implementation of a fitness proportional (roulette wheel) selection by soft-maxing the fitness scores and using those as probabilities to draw independent samples with replacement from the population.

3.1.4 Recombiner

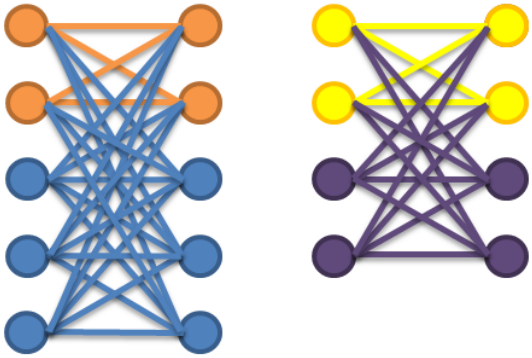
A recombiner takes care of recombining selected pairs from the mating pool. In order to reduce run time, we decided to use a form of weight-inheritance that keeps trained problem representations in the population. Additionally, we wanted the evolutionary algorithm to converge to networks of a size that fits with the compression measure integrated in the fitness function. A recombination strategy that produces children of random shapes would prevent this convergence. Therefore, the parents are layer-wise combined into two children of the same shapes, meaning that weights are recombined but inherited and convergence of network layer shapes can still occur.

The recombination strategy that we implemented is based on the single cut-off crossover proposed by [7]. For each layer of the parent networks, a crossover point is randomly determined. The layer matrices are then recombined according to the scheme in Figure 1. This process is repeated for every layer of the parent networks.

This approach leads to a constraint on possible network architectures, namely that networks are required to have the same number of neurons in each hidden layer. Relaxing this constraint would mean that neurons could be recombined into networks that require more or less outgoing connections. This scenario is problematic, as weights would then be removed or created from scratch. A similar argu-



(a) Matrix representation of weight inheritance in the recombination step.



(b) Network representation of weight inheritance in the recombination step.

Figure 1: Visualisation scheme for the recombination of a single layer from two parent networks, consisting of 4 and 5 neurons respectively. The cut-off point for this recombination example is 2.

ment holds for the constraint of a single cut-off point for both networks. If the cut-off points would be different for the two parent network layers, neurons would end up with too many or too few connections.

3.1.5 Mutator

During a generation each network has a certain chance to mutate, which is given by the mutation rate. If a network is chosen for mutation, the weights for each hidden layer will be multiplied by an uniformly drawn factor between 0.9 and 1.1. Note here that each weight has its own independently drawn factor.

3.2 Experiments

As stated in section 3.1.2 on fitness calculation, the fitness function is three-part with an associated weight for each part. These weights can be used to simulate different scenarios for experimentation. Most importantly, we will investigate the effects of implicit retraining by omitting our similarity measure (i.e. setting its weight to 0) and comparing it to scenarios where the measure is included. Excluding the case of all measures having 0 weight, there are thus seven scenarios possible as shown in Table 1. In our experimental settings, the similarity weight measure as described above is set to 2 to counteract effects of measure magnitude.

w_a	w_s	w_c	Interpretation
0	0	1	Only compression matters
0	2	0	Only similarity matters
1	0	0	Only accuracy matters
0	2	1	Accuracy does not matter
1	2	0	Network size does not matter
1	0	1	Similarity does not matter
1	2	1	All metrics matter equally

Table 1: Table of weight parameter configurations and their semantic meaning for the fitness function, w_a = accuracy measure weight, w_s = similarity measure weight, w_c = compression measure weight

We create an initial population of 100 individuals, with the initialisation method described in section 3.1.1. Due to the way we implement inheritance, the population size stays constant at 100 (see section 3.1.3). The mutation rate was set at 1%. As data for the evolutionary optimisation, we chose a balanced subset of 10000 images of the FashionMNIST dataset [9].

To avoid drawing false conclusion due to the inherent randomness of the algorithm, we run each each experiment 10 times. In each run, we train the population for one epoch (i.e. one pass through the training data), since we observed sufficient convergence behaviour with this amount of data. Each batch we pass corresponds to one generation in the evolutionary algorithm, including the selection, mating, mutation and evaluation of networks. The data amounts to 100 batches, or generations, with 100 datapoints each.

3.2.1 Parent Network and Data

All of these experiments were performed using the same network to compress as input. This parent network is a two-layer network, each layer has 256 nodes with ReLU activation functions. It was trained on the FashionMNIST dataset for 4 epochs of 600 batches with a batch size of 100. Measuring its per-

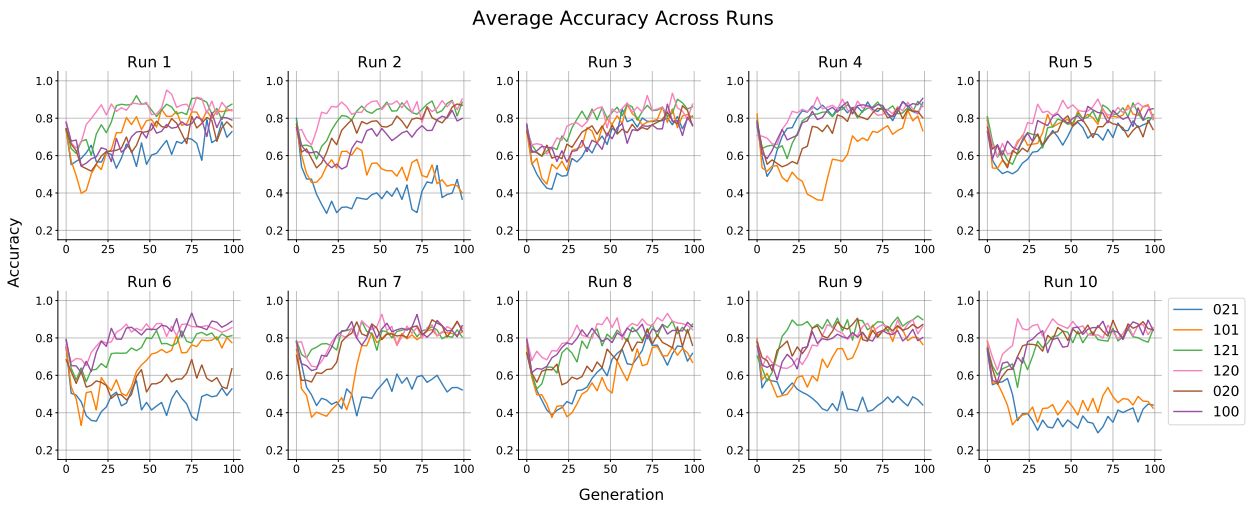


Figure 2: Average accuracy across the population networks reported every third generation. Higher is better. The legend numbers indicate the weights given to the components of the fitness function in the following order: accuracy, similarity, and compression loss.

formance on the validation set, it achieved an accuracy of 86.6%. The trained weights for this network are present in our repository on GitHub.

4 Results

The results consist of two parts. The first part describes the result on the main experiments as outlined earlier, where we investigated the effects of the different fitness function components. It reports average population metrics over time (reported per 3 generations) during the runs we performed. In the second part we will describe the characteristics of the final best networks EComp returned per run per experiment.

However, we omitted the experimental data for the case where only compression was used in the fitness function as its results turned out uninformative for answering our research question. As one would expect, networks produced by a compression-only fitness function converge quickly to a population of very small networks. Therefore, these networks suffered from low accuracies, rendering this setup useless for balanced neural network compression.

4.1 Main Experiment Results

For the main experiments we report the average accuracy, compression measure and the similarity loss per experimental scenario.

4.1.1 Accuracy

The first thing that stands out from the accuracy plots (Figure 2) is how often the experiments' populations converge to a certain average accuracy after around 30-50 generations, except for some runs. On further inspection, there is a clear distinction between consistent and inconsistent experiments, i.e. a low or a high variance in performance respectively. The setting that only cares about similarity and compression and the setting that does not care about similarity (indicated by the labels '021' and '101' in the plots below respectively) seem to vary the most, because they sometimes converge on a higher and more often than not on a lower accuracy. Next in terms of consistency comes the configuration that only cares about similarity (indicated by '020'). Except for a few runs it shows consistent and high performance, along with the configurations that optimise all characteristics ('121'), only accuracy ('100') and both accuracy and similarity but not compression ('120').

4.1.2 Compression

Figure 3 shows the results of the experiments in terms of the compression loss. Note that a lower value indicates stronger compression, and is thus better. We can again observe a similar convergence trend for most experimental conditions, which generally converge between 30-50 generations. In the previous section, the two conditions for which either accuracy or similarity were not integrated in the fitness function (labels '021' and '101', respectively) seemed to be exempt from such predictable convergence behaviour.

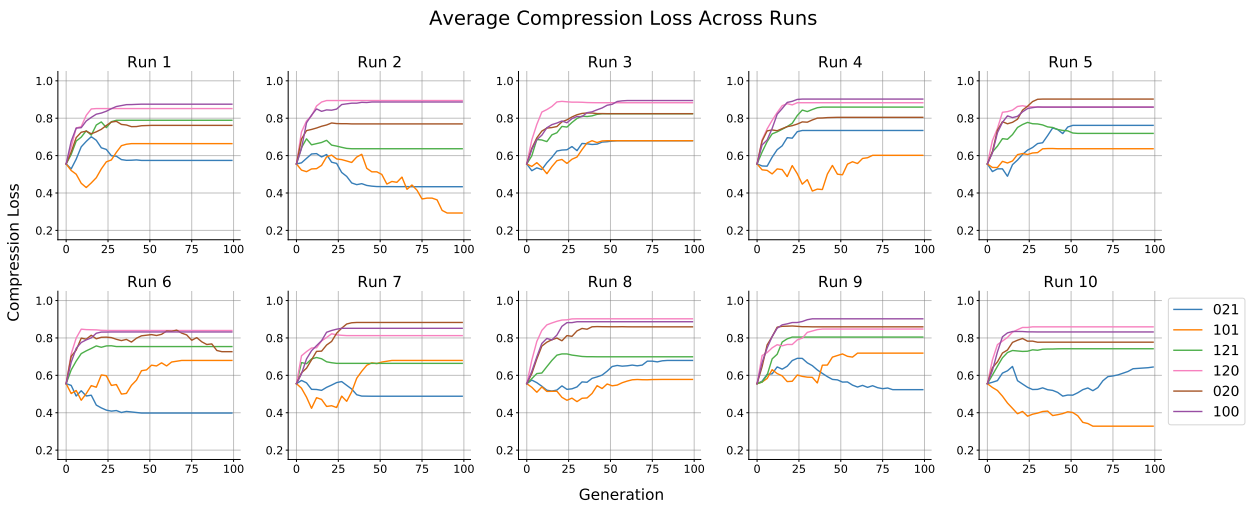


Figure 3: Average relative size with respect to the parent network across the population networks reported every third generation. Lower is better. The legend numbers indicate the weights given to the components of the fitness function in the following order: accuracy, similarity, and compression loss.

On the compression loss however, we observe the very same conditions achieving the best performance, with the '021' condition in 4 out of 10 runs, the '101' condition in 5 out of 10 runs, and one run with identical performance. In terms of both consistency and performance they are most often followed by the configuration that uses all three measures (label '121'). The remaining configuration, none of which use compression in their fitness determination, perform consistently the worst in this respect (labels '120', '020' and '100').

4.1.3 Similarity Loss

Figure 4 shows the results of the experiments in terms of the similarity loss. On first glance, this exhibit the highest variance, or noise, across runs, as compared to the two other metrics. For example, in some runs there is no clear distinction and the loss trends over the generations all follow the same pattern. This is the most prominent in the fifth runs but also to a lesser extent in runs 3 and 4 and an even lesser extent in 6 and 8. Still, certain patterns seem to emerge, like how trends seem to convergence roughly between generations 30-50 if they converge properly at all. Moreover, the configurations that do not explicitly optimise for accuracy or for similarity but do for the other measures (indicated by the labels '021' and '101' respectively) seem to perform the worst overall. On the contrary, two of the subset of the configurations labelled by '121' (all measures are taken into account for fitness), '020' (only embedding loss is used to calculate fitness) and '120' (all measures but compression are optimised for) seem to be nearly always present in the top 3 (not taking into account the conglomerate of run 5). The configuration that focused purely on accuracy ('100') seems to be the most inconsistent throughout the runs and performed both well and poor w.r.t the embedding loss between the runs.

4.2 Best Network Results

Another important outcome of EComp is the network that had the highest fitness in the final population. The boxplots in Figure 5 showcase the accuracy and the compression loss of the final networks of each experimental condition, aggregated over the 10 times every condition was executed for.

In the left plot we see that, whenever the accuracy does not matter for the fitness, there is a large difference in whether or not the the compression measure is used in combination with the similarity measure (the experiments labeled '021' and '020' respectively). In the case without compression ('020') the variance between the accuracy seems way smaller and on average higher (even though it does still have an outlier). We also see two outliers in the case labelled '101' where both the accuracy and compression matter for fitness but not the similarity and are in that sense very similar to the '020' experiment. The accuracy for the remaining three experiments seems high and consistent, competitive with the parent network and sometimes seemingly even outperforming it.

Figure 4: Average similarity loss across the population networks reported every third generation. Lower is better. The legend numbers indicate the weights given to the components of the fitness function in the following order: accuracy, similarity, and compression loss.

In the right plot, which depicts the compression loss (lower is better), we see that again the '021' experiment has the largest variance but is now performing the best on average, even though it seems very competitive with the '101' experiment that again has two outliers. Upon further inspection, the two outliers for the '101' experiment in both plots are the same two networks. The '020' experiment does not have any outliers in this plot and resulted in high compression loss. Only the consistent '120' and '100' experiments (who both do not use compression in their fitness functions) resulted in an ever higher compression loss. The last remaining experiment '121', that uses all three characteristics to determine fitness, seems to be inconsistent but performing relatively well.

5 Discussion

We started this work with the question whether adding a similarity measure to a neural network compressor based on an evolutionary architecture would be beneficial. We hypothesised that it would act as a suitable heuristic to maintain relevant information in the compressed network's embeddings, while pruning irrelevant information. Our results seem to support this hypothesis: as Figure 5 shows the networks trained with a fitness dependent on all three components are the most consistent in maintaining high accuracy while achieving good compression. When looking at the average population results in Figure 2, this experimental configuration was also one of the most consistent in reaching good performance. While other combinations of measures also reached decent results, they were less consistent and affected more by the random nature of the algorithm. For example the set-up with only accuracy and compression, but no similarity, also performed reasonably well albeit far less consistent. This can be interpreted as evidence for a regularising effect of the similarity measure. The two outliers that are present for this experiment in both the accuracy- and compression boxplots (Figure 5) imply a trade-off effect between accuracy or compression that can occur when not using similarity.

Another indication of this regularising effect is our observation that including the similarity loss in the fitness function decreases the variation in final accuracy between the runs, especially under conditions that do not optimise for accuracy. Without optimising for similarity these experiments were more affected by the algorithm's inherent random effects, causing larger variation. This may imply that accuracy and similarity are correlated, possibly because well performing subnetworks of the parent are approximated by the child. However, using just similarity without accuracy lowers the performance of the resulting networks. This may mean that the similarity can be attributed to regions in the embedding space that are uninformative, rather than those important for the actual model accuracy. We then observe that the inclusion of a compression component can negatively affect accuracy, and the positive contribution of similarity to accuracy in particular. Without compression as a fitness compo-

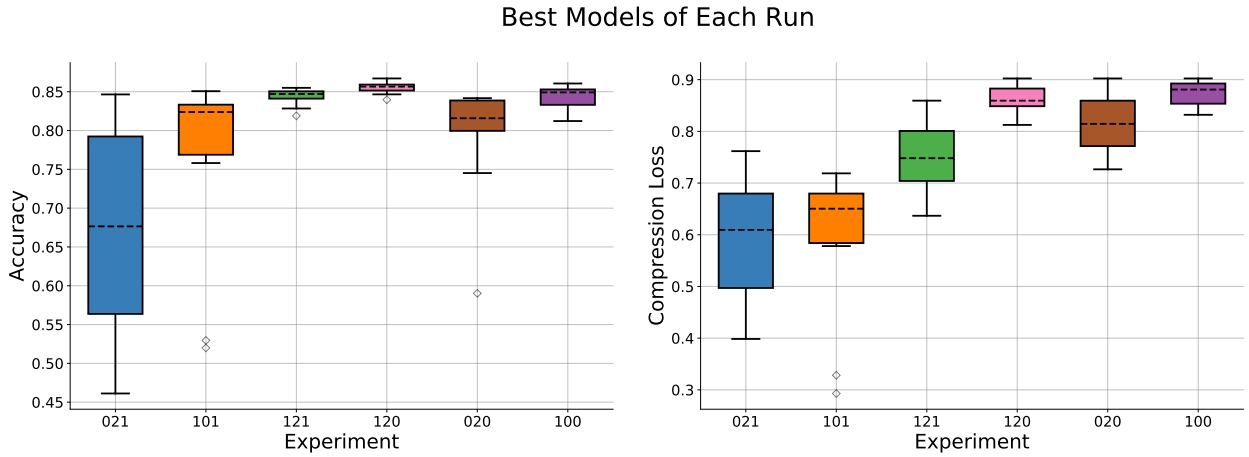


Figure 5: Box plots representing the collection of best networks (in terms of fitness) aggregated over 10 runs per experiment. The x-axis values indicate the weighting for the different components of the fitness function during an experiment, in the following order: accuracy, similarity, and compression loss.

nent however there seems to be no inherent drive to compress the networks, making it a necessary fitness component.

When looking at the similarity loss, we observed that its range of values was much lower than we expected. Preliminary experimentation indicated that using a weight of 2 for the similarity loss would give us roughly equal contribution of all the three measures to the overall fitness, but the numbers in the y-axes show that this scalar factor was not high enough. The similarity magnitudes were often within a range of 0.05 to a maximum of 0.4, with the best ones converging to a loss of around 0.1. Thus, most often this similarity measure would be prioritised less than the other two measures and we should have used a higher weight.

The different batches per run also showed us another interesting effect: the average population graphs of the accuracy and the similarity (Figs 2 and 4) show very similar batch-specific effects. For example run 3 and 5 are very similar for both measures: all the experiments seem to clump together. To a lesser extent more of these similarities can be found, such as runs 10 and 4. These similarities do not translate to the compression plot, which further strengthens the belief that similarity and accuracy are correlated.

As a practical remark we note that our EComp algorithm only works on and with balanced networks, which is a very serious limitation. Finally, we were surprised to see that EComp seemed to be relatively fast for our experiments. Without GPU acceleration (i.e. running on CPU) all the experiments could still be ran within a matter of 2 minute per run on an ordinary desktop computer using an Intel i7-6700K CPU with 32 GB of DDR4 memory. However, we only used relatively small networks and a population size of 100 for our experiments. Therefore, it is unclear how well EComp scales up to larger networks and populations.

6 Conclusion

We presented EComp: an algorithm to compress a fully trained balanced network using an evolutionary approach. It is novel in the sense that it adds a measure for similarity between the parent and children networks in the fitness function, next to accuracy and compression measures. Our proposed addition is related to the 'lottery ticket' hypothesis: in order to effectively compress a network, the algorithm should only keep the most relevant weights and subnetworks. In this report, we found experimental evidence that adding the similarity measure to the accuracy and compression measures guides the algorithm to do exactly this.

These findings show that Ecomp is a promising approach to neural network compression, and therefore offers a foundation for future research to build upon. For example, it would be fruitful to investigate how it holds up against current state-of-the-art compressors, that may or may not use retraining, in performance in terms of its compression qualities and run time. Since we only experimented with a limited number of weight configurations for the loss measures, it would be further worthwhile to

determine the optimal weights to be used in future versions of the algorithm. Finally, making EComp work for unbalanced networks of all shapes and sizes is also an unsolved challenge for the future. We hope that this report will inspire more work on these approaches and most of all more experimentation, adaptation and extension of EComp.

7 Contributions

Pascal	Stijn	Thijme
T-SNE research	EA skeleton	initialiser
similarity loss	selector	recombiner
code fixer	mutator	inheritance
report	main writer	report

Table 2: Contributions

References

[1] G. Fang, J. Zhang, and Z. Kong. Torch-pruning. https://github.com/VainF/Torch-Pruning, 2021.

[2] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[3] J. Huang, W. Sun, and L. Huang. Deep neural networks compression learning based on multiobjective evolutionary algorithms. *Neurocomputing*, 378:260–269, 2020.

[4] Y. Liu, Y. Sun, B. Xue, M. Zhang, and G. Yen. A survey on evolutionary neural architecture search. *arXiv preprint arXiv:2008.10937*, 2020.

[5] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pages 293–312. Elsevier, 2019.

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[7] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics*, 50(9):3840–3854, 2020.

[8] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[9] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[10] Y. Zhou, G. G. Yen, and Z. Yi. Evolutionary compression of deep neural networks for biomedical image segmentation. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):2916–2929, 2020.

[11] Y. Zhou, G. G. Yen, and Z. Yi. A knee-guided evolutionary algorithm for compressing deep neural networks. *IEEE Transactions on Cybernetics*, 51(3):1626–1638, 2021.