Evolutionary Approaches for OOD Ensembles

Scott T. Merrill
Department of Computer Science
University of North Carolina
smerrill@unc.edu

1 Introduction

Traditional Machine Learning (ML) models are trained under the assumption that the training data and testing data follow the same distribution. However, such assumption might not hold in practice. The Out-of-Distribution (OOD) generalization task aims to train trustworthy worthy models under any hypothetical distribution shift. One approach to this problem may be to ensemble a diverse collection of models. Given each individual model is reasonably accurate and produces independent errors with respect to the other models, mistakes may cancel out leading to an ensemble with good generalization ability Johansson et al. [2007]. This paper explores several approaches based on Evolutionary Algorithms (EA) to construct ensembles for OOD generalization. In total, four algorithms are considered which can be broadly categorized based on the evolutionary fitness function they are optimizing. Two algorithms attempt to optimize for Quality Diversity (QD) Pugh et al. [2016]; that is, models that are both accurate and novel. The second class of algorithms attempts to optimize a fitness function to approximate the true OOD accuracy. We compare the two approaches and comment on the appropriateness of each objective for OOD generalization.

2 Related Work

2.1 Neural Architecture Search

Neural architecture search (NAS) treats the architecture of a NN as a hyper-parameter and attempts to automate the network design. NAS can be mathematically formulated as an optimization problem:

$$A^\star = \arg\min_{A \in \mathcal{A}} L\left(A, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{fitness}}\right) \tag{1}$$

where $\mathcal{A}$ is the search space of neural architectures and $L\left(\cdot\right)$ is a loss function that evaluates the performance of an architecture A on a fitness evaluation dataset $\mathcal{D}_{\text{fitness}}$ after being optimized on the training set $\mathcal{D}_{\text{train}}$ Liu et al. [2021].

The NAS optimization problem described in Equation 1 is typically solved with one of the three approaches: reinforcement learning, gradient-based search, and evolutionary-based methods. Among which, we focus our discussion on evolutionary methods that apply genetic operations such as mutation and cross-over to potential solutions in the search space. These methods do not require gradient information and are therefore less susceptible to local minima and particularly well-suited for optimization problems with an objective function that does not have a closed-form expression Darwish et al. [2020].

2.2 NAS-OOD

Few have attempted NAS strategies for out-of-distribution (OOD) generalization. To the best of our knowledge, no evolutionary NAS strategies have been applied to the OOD generalization problem. One major challenge using NAS is that NN could easily overfit to training data as both the architecture and the weights are optimized simultaneously; this leaves the learned networks particularly vulnerable

Preprint. Work in progress.

to distribution shifts. Bai et al. proposed NAS-OoD using a conditional data generator to generate synthetic OOD data and further use it to guide NAS Bai et al. [2021]. However, NAS-OoD not only assumes knowledge about the context of each instance but also relies on gradient to perform the search, which is susceptible to local minima. Our method does not require knowledge about the context and does not require gradient to perform the search.

2.3 Ensemble of Averages (EoA)

While traditionally the final NN weights found through gradient descent are used, Arpit et al. [2022] take an alternative approach of calculating the moving average of network parameters over the optimization trajectory. EoA ensembles together several moving-average weights collected from networks with different hyper-parameters and initializations. The authors show that ensembling these moving average models is beneficial for OOD generalization. We highlight EoA as we see many parallels with our approach of combining all of the models in the final generation. Thus, in a similar fashion, we are averaging over the optimization trajectory of an EA.

2.4 Sparsely-Gated Mixture of Experts

In ensemble methods, the predictions made by each expert are often averaged. Similar to ensemble methods, mixture of experts (MoE or ME) model also maintain a collection of models. The difference is that MoE assumes each individual expert specializes in different "aspect" of the task we want to solve; hence, the output of individual experts should be mixed "differently" depending on the current instance instead of weighing expert opinions uniformly as in ensemble methods. Shazeer et al. proposed a Sparsely-Gated MoE (SMoE) Layer that has a learnable gating function that generates a sparsely distributed mixing coefficient to encourage specialties Shazeer et al. [2017]. This designed sparsity allows massive expert models to be trained efficiently.

3 Methods

We next discuss our approach to construct evolutionary ensembles for OOD prediction. We first describe two algorithms that use QD metrics to identify diverse populations of NN architectures. We then discuss an approach which attempts to construct a fitness function to approximate a models true OOD performance.

3.1 Quality Diversity Approaches

In this section we show how to evolve the number of hidden units, hidden layers and feature subsets of a population of NNs that produce predictions with high novelty and accuracy.

3.1.1 Genetic Encoding and Operators

To evolve a population of NNs using GA, we must define the space to be evolved, and the operations which will be applied to evolve the space. Each network, n_p in a population is encoded by a layer vector $L = [l_1, ..., l_i]$ and a feature vector $F = [f_1, ..., f_d]$. The value l_i in the layer vector defines the number of nodes in the ith hidden layer of the network. Thus, the size of L governs the number of layers in the network. In contrast, the size of F is of fixed length equal to the number of features in our dataset; each f_i corresponds to a binary variable indicating whether to use feature i when training the network. Moreover, the input size of network n_p, will have an input size equivalent to the number of 1s in F.

Network encoding are evolved using only mutation operations. Mutations are easily implemented, offer a naturally divergent search, and avoid any assumptions that similar network encodings should behave similarly. A mutation, as we define them, performs one of four operations with equal probability.

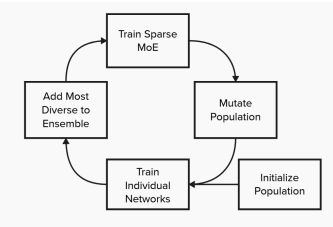
Insert a layer Adds a layer to a random position in the network. The number of nodes in this new layer will also be randomly specified by hyperparameters dictating the minimum and maximum nodes allowed in a layer.

Swap a layer Swaps two layers selected randomly.

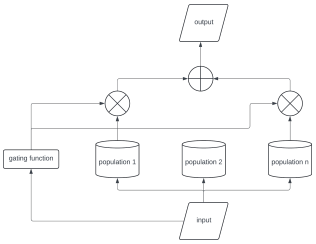
Remove a layer Remove a layer selected at random.

Randomize the feature encoding Generates a new binary feature encoding at random.

To prevent networks from becoming too deep or too wide, the insert and remove operations are controlled by hyperparameters that restrict the maximum number of layers and allowable nodes in each layer.



(a) An overview of our proposed evolutionary OOD ensembles (EVO-OOD) method.



(b) An overview of our proposed co-evolutionary OOD ensembles (CoEVO-OOD) method.

Figure 1: Overview for both of our proposed evolutionary ensemble algorithms.

3.1.2 Evolutionary NaS OOD Ensembles (Evo-NaS)

Figure 1a summarizes our first approach which we call, Evo-NaS. We first randomly initialize a set of $P = [n_i, ..., n_p]$ network encodings. Due to the computation limitations of training each network in the population on every evolutionary generation, we train each network for a just a few epochs to get an idea of the solution it's learning. After each network is trained, the fitness of each network is computed as a function of its accuracy and novelty with respect to the existing ensemble. These methods are computed differently for regression and classification tasks as described below.

$prop_{i,j}$ = $(N^{01} + N^{10})$ / $(N^{00} + N^{01} + N^{10})$

Classification Task: $fitness_{ij}$ = $accuracy + prop_{i,j}$

Regression Task: $fitness_{ij}$ = $r^2 + cosdist_{ij}$

Where $prop_{i,j}$ is the proportion of error differences between classifiers i and j. N^{01} implies classifier i made an incorrect prediction while classifier j made a correct prediction. And $cosdist_{ij}$ refers to the cosine distance between two prediction vectors. Note that in both tasks, fitness encourages high accuracy and diverse errors with respect to other classifiers. The NN with the highest fitness is then added to the ensemble and the ensemble is trained using SMoE. The designed sparsity of this model enables a large collection of models to be trained more efficiently than if they were trained individually. The current population is then mutated and the next generation begins. This process is repeated for a fixed number of generations. Note the number of generations is a hyperparameter that directly governs the number of experts included in the model.

3.1.3 Co-Evolutionary NaS OOD Ensembles (CoEvo-NaS)

Figure 1b demonstrates our second approach, which we call CoEvo-NaS. The premise is to co-evolve m distinct populations. Each population will evolve NNs with different characteristics. One population, for example, may be evolving NNs with a certain amount of noise added to the features and a Tanh activation function while another population may only be evolving a population of NNs with no noise that strictly use RelUs. Like Evo-NaS, in each population, we evolve the number of units, layers and feature subsets. The idea is to then ensemble the single best model in each population.

CoEvo-NaS begins by randomly initializing a set of encodings in each population. Each network in each population is pre-trained and the network with the highest fitness with respect to the existing ensemble on the prior generation are aggregated and trained using SMoE. Finally, each population is mutated. This process is repeated for a fixed number of generations. Unlike Evo-NaS, the number of experts in CoEvo-NaS is controlled by the number of distinct populations, not the number of generations of evolution run.

3.2 OOD Approximation Approaches

In this section we will explain two methods to evolve ensembles of XGB Trees Chen and Guestrin [2016] for OOD prediction tasks. Evolution will identify optimal subsets of data and subset of features used to train each individual model. A custom fitness function which judges how well all the models work together in classifying OOD examples will be optimized.

3.2.1 Genetic Encoding and Operations

We define our genetic encoding and the evolutionary operators that modify these encodings. Note that our encodings rely on training examples being first pre-processed into distinct clusters. This pre-processing step can be achieved, for example, through kmeans, Decorr Liao et al. [2022] or any hard-clustering algorithm.

Encoding: Each XGB Tree model m is represented fully by two vectors; a cluster vector $C = [c_1, ..., c_k]$ and a feature vector $F = [f_1, ..., f_d]$. Each are binary indicator vectors where $c_i = 1$ implies the m was trained on cluster i and $f_j = 1$ indicates the model has access to feature j.

Mutation: Mutations operate the same on both the cluster and feature vector. We define two mutation operators. The first selects n elements from one of these vectors as candidates for mutation. These n indices are each flipped independently with probability p. We also consider a simpler operation where all k clusters and all d features are considered candidates for mutations and flipped with probabilities p_k and p_d respectively.

We perform mutations over either clusters or features but never both simultaneously. The intuition behind this is it is easier to optimize one set of parameters at a time. Consider for example, a favorable mutation to the feature vector that is matched with an unfavorable mutation to the cluster vector. If the model's classification accuracy improves or degrades, it is impossible to isolate the source of improvement or degradation.

Crossover: We define two crossover methods which operate on ensembles of models. Each ensemble E contains a collection of N models each $[m_1, ..., m_n]$. The first crossover method selects a random integer i between 1 and n. Child 1 inherits a sample of i models from Parent 1 and a sample of the remaining $n - i$ models from Parent 2. Conversely, Child 2 inherits a sample of i models of Parent 2 and sample of the remaining $n - i$ models from Parent 1. We define a second crossover operation in which we select i random models in Parent 1 and swap them with i models in Parent 2. This operation thus requires an additional hyperparameter i representing the number of models to be swapped.

Fitness Function: We borrow an idea from training random forests and define the fitness function of our algorithm as the out-of-bag (OOB) error. The fitness of a model is defined by the average accuracy when predicting on clusters the model was not trained on. The fitness of an ensemble is a voting classifier of all models not trained on a particular cluster.

3.2.2 Evolutionary Out of Distribution Ensembles (Evo-OOD)

Evo-OOD in this section is similar to Evo-NaS with a few key differences. Instead of evolving the features, layers and nodes in each layer of a NN, we instead evolve the features and the data used to train the XGB Trees. Additionally, Evo-OOD attempts to optimize OOB error rather than maximize QD of the resulting ensemble. Evo-OOD begins by initializing a population of models. In each generation, the fitness of each model in the population is calculated according to it's OOB error. The best performing models will be candidates to add to the existing ensemble. Each of these candidates are temporarily added to the existing ensemble for the purpose of calculating the OOB error of the ensemble when that candidate is included. After all candidates are tested, the single candidate that results in the best OOB error of the ensemble is added. Finally, before starting the next generation, the best models are mutated and replace the worst performing models. Similar to Evo-NaS, a single model is added to the ensemble each generation. Thus the total ensemble size corresponds to the number of generations of evolution.

3.2.3 Co-Evolutionary Out of Distribution Ensembles (CoEvo-OOD)

CoEvo-OOD in this section is similar to CoEvo-NaS; several populations are co-evolved and an ensemble is constructed by selecting one member from each population. Specifically, we first initialize

M distinct populations, each containing N models. Each population, corresponds to a particular cluster encoding; that is, all the models in a population are trained on the same clusters of data and only the feature sets are being evolved within the populations. In every generation, the fitness of all $M * N$ models is first calculated. We next sample one model from each population based on a temperature scaled softmax of the population fitness vector to form an ensemble. We perform this sample operation multiple times to construct multiple candidate ensembles and compute their accuracy metrics. Before beginning the next generation, we mutate the features of the best models in each population and replace the least fit individuals. Note that we are selecting one model from each population to be included in the ensemble and which enforces cluster diversity. By incorporating this domain knowledge that cluster diversity is important we can reduce the search space of evolution to only search the space of features allowing evolution to operate more efficiently. And, by comparing CoEvo-OOD to Evo-OOD we can comment on the extent to which reducing the search space helped guide evolution to an efficient solution.

4 Experiments

4.1 Real-World Dataset

We consider adult [Becker and Kohavi, 1996] dataset from UCI machine learning repository. We arbitrarily choose one of the predictor variables and make in-distribution and out-of-distribution sets. Instances with attribute "workclass" equal to "federal-gov", "local-gov", "state-gov" are considered as out-of-distribution and the rest being in-distribution. The in-distribution dataset is split into training set, validation in-distribution set, and test in-distribution set following the 80/10/10 splitting ratio. The out-of-distribution dataset is split into validation out-of-distribution set and test out-of-distribution set with 50/50 ratio.

4.2 Results

4.2.1 Quality Diversity Approaches

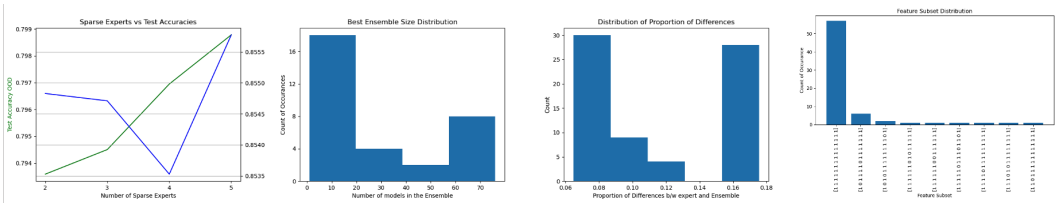


Figure 2: Analyzing Model Diversity

We ran Evo-NaS and CoEvo-NaS with a population size of 100 for 100 generations. We allowed evolution to evolve networks of size at most 5 with at most 250 hidden units in each layer. Networks were required to use at least one feature; no other restrictions were placed on feature subset choices. We ran both models setting the number of sparse experts to use in a single prediction to 2, 3, 4 and 5. The best models for Evo-NaS and CoEvo-OOD are shown in Table 1. The notation Evo-NaS(l, n, f) implies mutation is performed on number of layers, number of nodes and feature subset.

We compare our model with three baselines for performance; a logistic regression, an EoA, and a MoE model with 100 experts, each being a single layer NN with 128 units. The latter of these baselines is of particular importance as it allows us to analyze the importance diversity in network structure can have on an MoE model. Our Evo-NaS models seem to consistently outperform the MoE baseline, though the differences are very marginal for this dataset. Similarly, EoA seems to have a slight advantage over our models. While our models have a much higher capacity, the drop in accuracy on validation and test sets is similar to that of other models. This is an encouraging sign indicating we are overfitting to the same degree as these other less complex models. Figure 2a, shows the Test ID and Test OOD results for Evo-NaS(l,n,f) while varying the number of sparse experts. The plot indicates little benefit can be obtained by routing examples to more experts, suggesting there is little diversity in the top-k expert predictions.

While we let Evo-NaS run for the full 100 generations, we additionally saved the model with the best validation accuracy through each generation. We aggregated all our runs of Evo-NaS and analyzed the distribution of the size of the ensemble resulting in the best validation accuracy. As shown in Figure 2b, in a large number of trials, the best ensemble is found after only a few iterations. This is yet another indication of a limited benefit from diverse expert predictors.

To further study the diversity of our ensembles, we investigate a single run of Evo-NaS(l, n, f). We selected the model with the largest ensemble size; this model resulted 71 individual expert models. We compute the proportion of differences between each expert model's predictions and the predictions of the entire ensemble and show the distribution of proportion of differences in Figure 2c. We also show the distribution of feature encodings of each of the 71 experts in Figure 2d. Several more facts highlight limited benefits achieved from our ensemble's diversity. First, we realize the validation accuracy of the ensemble is 85.02% while the validation accuracy of the single best expert is 84.47%. Second, while there are 2^{14} permutations for the featuring encoding vector, the resulting ensemble contains only 9 of these permutations. Furthermore, most of the ensemble members use all 14 features. Finally, the proportional of errors distribution is bimodal. This last point is a bit concerning and potentially indicates mutation often finds models that make really unique errors. Moreover, our evaluation of fitness prioritizes these extremely novel models which may not be very accurate. Diversity is good insofar as it improves ensemble accuracy. Furthermore, prioritizing it too much at the expense of ensemble accuracy could pose some problems.

4.2.2 OOD Approximation Approaches

We next analyze the performance of the two methods that explicitly attempt to optimize OOB fitness. We ran both models for 50 generations with 10, 30 and 50 kmeans clusters. Additionally for CoEvo-OOD we set the number of populations to 10, 20 and 30. The cluster encodings in each population were initialized randomly to be of a certain size (ie. to have a certain number of bits set to 1); three specific sizes were tested. This design decision was based on analysis that limiting the cluster encoding size appears to result in an OOB fitness function that relates more strongly to true OOD accuracy.

We report the best Evo-OOD model in Table 1 and show the learning curve for this model in Figure 3. We also show a distribution of both the cluster and feature distributions contained in the final ensemble. We notice an initial spike in accuracy scores that deteriorates as the benefit of adding more predictors is diminished. Perhaps adding a metric to encourage diversity would delay this deterioration. We also notice that many of the models rely on a few informative clusters while other clusters are rarely used. An interesting direction of future work questions why these clusters are seldom selected; perhaps these are outlier instances and relying on them would corrupt predictions. We notice a similar pattern in the feature distribution; some features are selected by nearly all models in the ensemble while others are often overlooked. It would be interesting to relate the feature distribution found by optimizing our OOB fitness with the principle components of our data and other classical feature importance metrics in ML. We bookmark this analysis for future work.

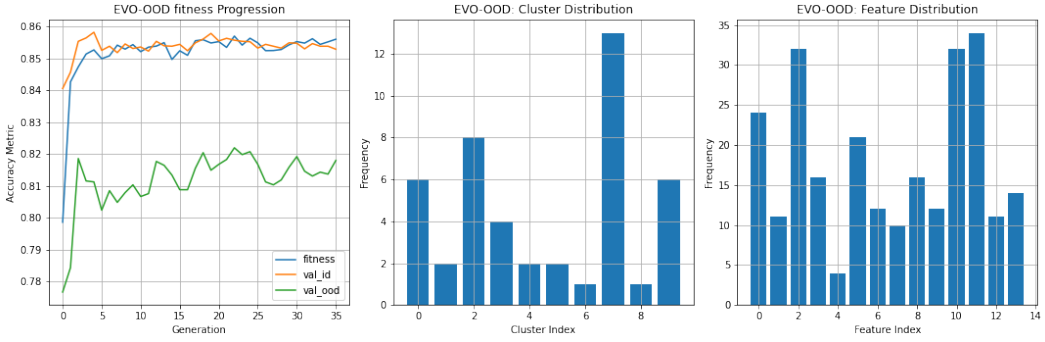


Figure 3: EVO-OOD learning curve and resulting cluster and feature distributions

We report the best CoEvo-OOD model in Table 1 and show it's learning curve in Figure 4. We ran CoEvo-OOD in two scenarios; where each co-evolving population uses a distinct cluster encoding and where these populations have distinct feature encodings. In the former, evolution operates over

the features and in the latter, cluster encodings are optimized. We call the first case CoEvo-OOD(f) since only the features are evolving and the second case CoEvo-OOD(c) since only the clusters are evolving. In both scenarios, the best performing models were found using 10 kmeans clusters. For CoEvo-OOD(c), the optimal initialization of the fixed feature encodings were found to contain between 11 and 13 features. For CoEvo-OOD(f), the fixed cluster encodings that gave the best result used at most half of the clusters. CoEvo-OOD(c) produced higher accuracy's which might indicate varying the training data of XGB models has a greater effect on OOB fitness than varying the features. We also note that all algorithms which optimize for this OOB fitness appear to dominate those that optimize for QD metrics. We believe this isn't a function of the models we considered but speaks to the value of our proposed fitness objective; optimizing this metric results in ensembles more aligned with true OOD generalization task than optimizing ensembles based on QD metrics.

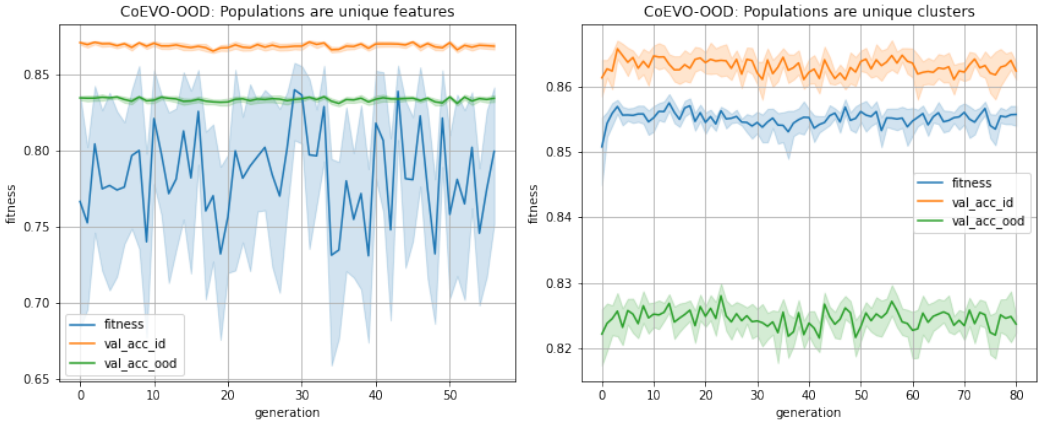


Figure 4: CoEvo-OOD learning curve and resulting cluster and feature distributions

From the learning curves we notice that when features are fixed and the clusters evolving, OOB fitness is considerably more volatile. This implies changing the trainin data alone can have a large impact on model performance. Interestingly, when clusters are fixed, model performance is only marginally affected by selecting alternative feature sets. This is another indication that allocating a greater number of resources to the evolution training clusters might be beneficial.

We were interested in better understanding the interaction effect between feature and cluster subsets. Specifically, we hypothesize that given certain clusters, there exists a set of features that optimizes OOB fitness (and presumably OOD accuracy). And similarly, given fixed features, we believe there exists a set of clusters that optimize our fitness function. We design a simple experiment to understand whether these hypothesis' hold and learn how difficult it is to find these optimal feature and cluster subsets. Specifically, we initialized three populations of size 50. Two populations had cluster encodings that differed by only a single bit, and the third population had a cluster encoding that was markedly different than the first two populations. We evolved only the features of each population using our mutation operations and OOB fitness function. We additionally design a second experiment in a similar way except it's the features that were fixed and the clusters that are being evolved. Figure 5 shows the results where we fix the features in each population and evolve the clusters. Figure 6 shows the results where clusters are fixed and features are evolved.

We notice, in each case, the optimal features for a given cluster can be found in just a few generations. Additionally, similar cluster encodings result in feature subsets distributions that are also similar. Furthermore, clusters with very different encodings have a very different optimal feature distributions. We see the same behavior by fixing features and evolving the clusters.

From this study we can draw several conclusions. First there are several local optima of our OOB fitness. But more importantly, the local optima are close in genetic space. With OOB fitness displaying some continuity with respect to our genetic encoding, we should be able to optimize it with genetic operations. Furthermore, the population based search of EAs allow it to explore many different solutions simultaneously making it more robust to local optima than many other optimization procedures. For these reasons, we argue EAs are well suited to optimize this OOB fitness metric.

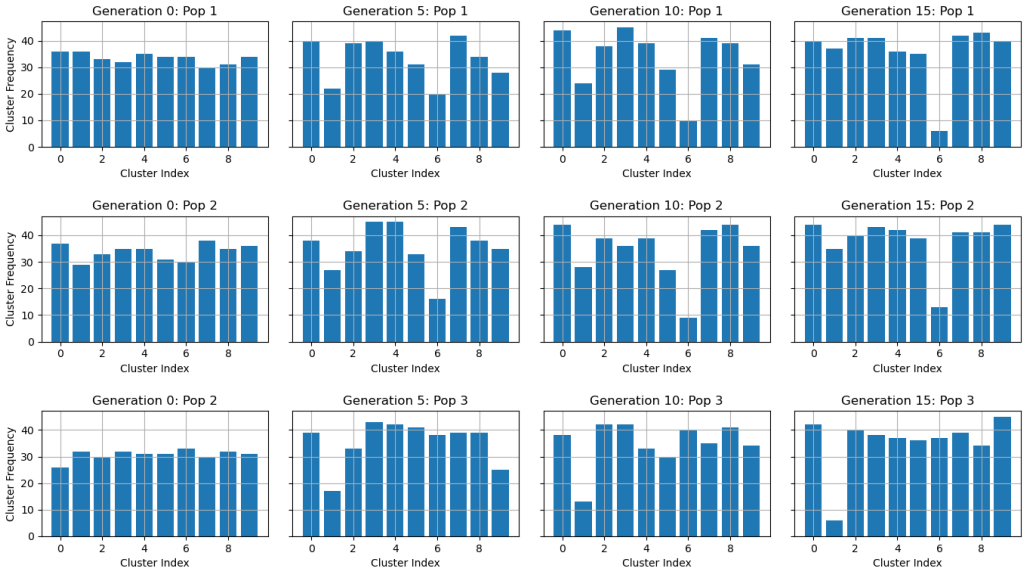


Figure 5: Results from our exploration of OOB fitness when features are fixed

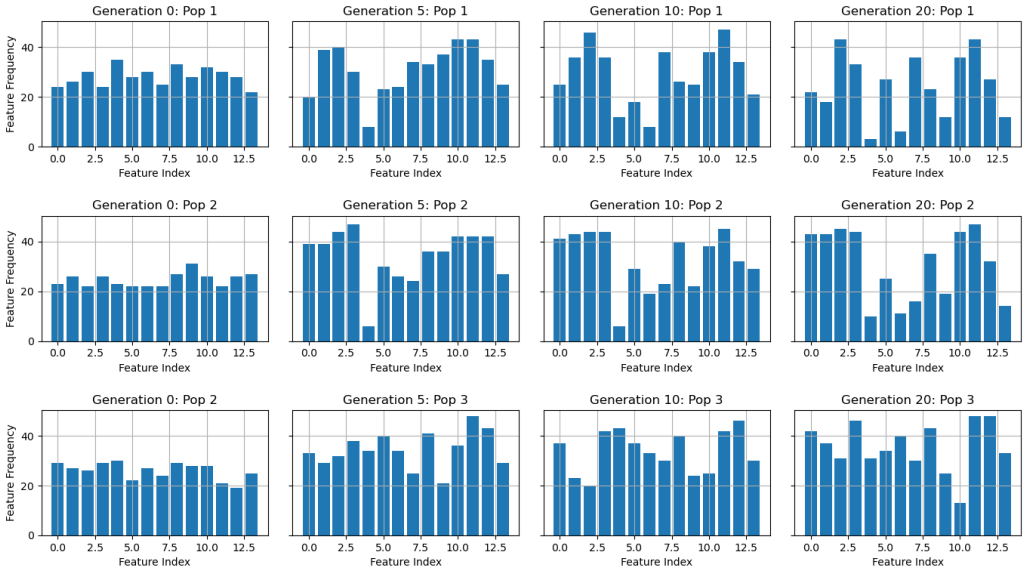


Figure 6: Results from our exploration of OOB fitness when clusters are fixed

5 Conclusion

We have explored several evolutionary to evolve ensembles specifically for OOD generalization tasks. Our approaches can be broadly categorized into QD approaches and OOD approximation approaches. The former attempts to evolve models that are accurate and produce independent errors, while the latter attempt to optimize a proxy of true OOD accuracy. We find that the OOD approximation approaches dominate the QD approaches, highlighting the value in our proposed approach of using OOB error as a proxy for true OOD accuracy. We provided a simple experiment to better understand this OOB fitness metric and find that while the function contains several local optima, it appears to be relatively continuous with respect to our genetic encoding. For this reason, we identify EAs as a viable approach to optimize this fitness function. While many ensemble approaches have been

Classifier	Train	Val-ID	Val-OOD
Baseline 1: Logistic Regression	83.32	82.64	77.79
Baseline 2: MoE	84.70	84.61	78.28
Baseline 3: EoA	85.65	85.55	80.66
Evo-NaS(l, n, f)	86.44	85.57	79.84
Evo-NaS(l, n)	**87.16**	85.50	79.44
CoEvo-NaS(l, n, f)	84.25	84.18	78.03
CoEvo-NaS (l, n)	84.10	83.73	77.91
Evo-OOD	85.70	86.81	83.66
CoEvo-OOD(f)	85.76	86.42	83.35
CoEvo-OOD(c)	85.85	**87.22**	**84.06**

Table 1: Accuracy Comparison on Adult Dataset

attempted for OOD generalization, to the best of our knowledge, this is the first work that applies EAs to evolve ensembles explicitly for OOD problems. We hope our work encourages more research into EAs to construct ensembles for OOD generalization.

References

Ulf Johansson, Tuve Lofstrom, and Lars Niklasson. The importance of diversity in neural network ensembles - an empirical investigation. In *2007 International Joint Conference on Neural Networks*, pages 661–666, 2007. doi: 10.1109/IJCNN.2007.4371035.

Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers Robotics AI*, 3:40, 2016. URL `https://api.semanticscholar.org/CorpusID:21713708`.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*, 34(2):550–570, 2021.

Ashraf Darwish, Aboul Ella Hassanien, and Swagatam Das. A survey of swarm and evolutionary computing approaches for deep learning. *Artificial intelligence review*, 53(3):1767–1812, 2020.

Haoyue Bai, Fengwei Zhou, Lanqing Hong, Nanyang Ye, S-H Gary Chan, and Zhenguo Li. Nas-ood: Neural architecture search for out-of-distribution generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8320–8329, 2021.

Devansh Arpit, Huan Wang, Yingbo Zhou, and Caiming Xiong. Ensemble of averages: Improving model selection and boosting performance in domain generalization. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=peZSbfNnBp4`.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=B1ckMDqlg`.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, August 2016. doi: 10.1145/2939672.2939785. URL `http://dx.doi.org/10.1145/2939672.2939785`.

Yufan Liao, Qi Wu, and Xing Yan. Decorr: Environment partitioning for invariant learning and ood generalization, 2022.

Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.