Evolution-inspired augmentations improve deep learning for regulatory genomics

Nicholas Keone Lee[1], Ziqi Tang[1], Shushan Toneyan[1], and Peter K Koo[1,*]

[1]Simons Center for Quantitative Biology, Cold Spring Harbor Laboratory, Cold Spring Harbor, NY, USA
[*]koo@cshl.edu

ABSTRACT

Data augmentations can greatly enhance the generalization of deep neural networks (DNNs). However, there are limited strategies available in regulatory genomics because modifying DNA can alter its function in unknown ways. Here we introduce a suite of evolution-inspired augmentations and circumvent issues of unknown function through a fine-tuning procedure using the original, unperturbed data. We demonstrate this approach improves generalization and model interpretability across several established DNNs in regulatory genomics.

Uncovering *cis*-regulatory elements and their coordinated interactions is a major goal of regulatory genomics. Deep neural networks (DNNs) offer a promising avenue to learn these genomic features *de novo* by being trained to take DNA sequence as input and predict regulatory functions[1–3]. Upon training, these models have been employed to score the functional effect of disease-associated variants[4,5]. Moreover, model interpretability has revealed that they base their decisions on learning motifs of transcription factor (TF) binding sites and their dependencies with other TFs and sequence context[6–10].

Empirically, generalization of DNNs improves with more training data. In functional genomics, the amount of data generated with high-throughput experiments is limited by the underlying biology. For example, the extent that certain TFs bind to DNA is constrained by the availability of high-affinity binding sites in accessible chromatin. An effective strategy to expand a finite dataset is data augmentations[11,12], which provides additional variations of existing training data. Data augmentations act as a form of regularization, guiding the learned function to be invariant to symmetries created by the data transformations[13,14]. This can impede a DNN's tendency to overfit on spurious features[15], leading to better generalization.

The main challenge with data augmentations is quantifying how the regulatory function under study changes for a given transformation. With image data, basic affine transformations can translate, magnify, or rotate an image without changing its label. For genomics, the available neutral augmentations are reverse-complement transformations[16] and small random translations of the input sequence[17,18]. With the finite size of experimental data and a paucity of augmentation methods, there exist only limited strategies to promote generalization for genomic DNNs.

Evolution provides a natural set of methodologies for creating genetic variability, with the purpose of broadening phenotypes to facilitate selection. Evolution-based perturbations include random nucleotide mutations, deletions, insertions, inversions, and translocations, among others[19]. Such data augmentations could generate larger shifts in sequence variation beyond the natural genome. However, it remains difficult to introduce neutral genetic perturbations that minimally impact the underlying biological function.

Here we introduce EvoAug, an open-source PyTorch package that provides a suite of evolution-inspired data augmentations (Fig. 1a). To address issues with unknown biological function, we employ a two-stage training procedure. First, a DNN is trained with evolution-based augmentations using the same training labels as the original wildtype sequence. This provides a modeling bias to learn invariances of the (un)natural symmetries created by the augmentations. For instance, random insertions and deletions impose a prior that motif distance is not important, while random inversions and translocations promote invariances to motif strand orientation and the order of motifs, respectively. However, biasing the learned function to be invariant to symmetry groups that do not accurately reflect the underlying biology may lead to poor generalization. Therefore, in the second stage, the DNN is fine-tuned on the original, unperturbed data (see Methods). In principle, pretraining with additional (albeit synthetic) genomic variation should facilitate learning more robust representations of features, such as motifs, while the fine-tuning stage should refine these features and guide the pretrained function towards the observed biology.

EvoAug randomly applies augmentations, individually or in combinations, online during training to each sequence in a mini-batch of data. Each augmentation is applied stochastically and controlled by hyperparameters intrinsic to each augmentation (see Methods). Optimal hyperparameter settings can be identified for each augmentation strategy by performing a hyperparameter sweep (Supplementary Figs. 1-5). In practice, these augmentations could disrupt core motifs and/or sequence context; however, they should not affect the sparse motif signals in non-coding DNA on average.

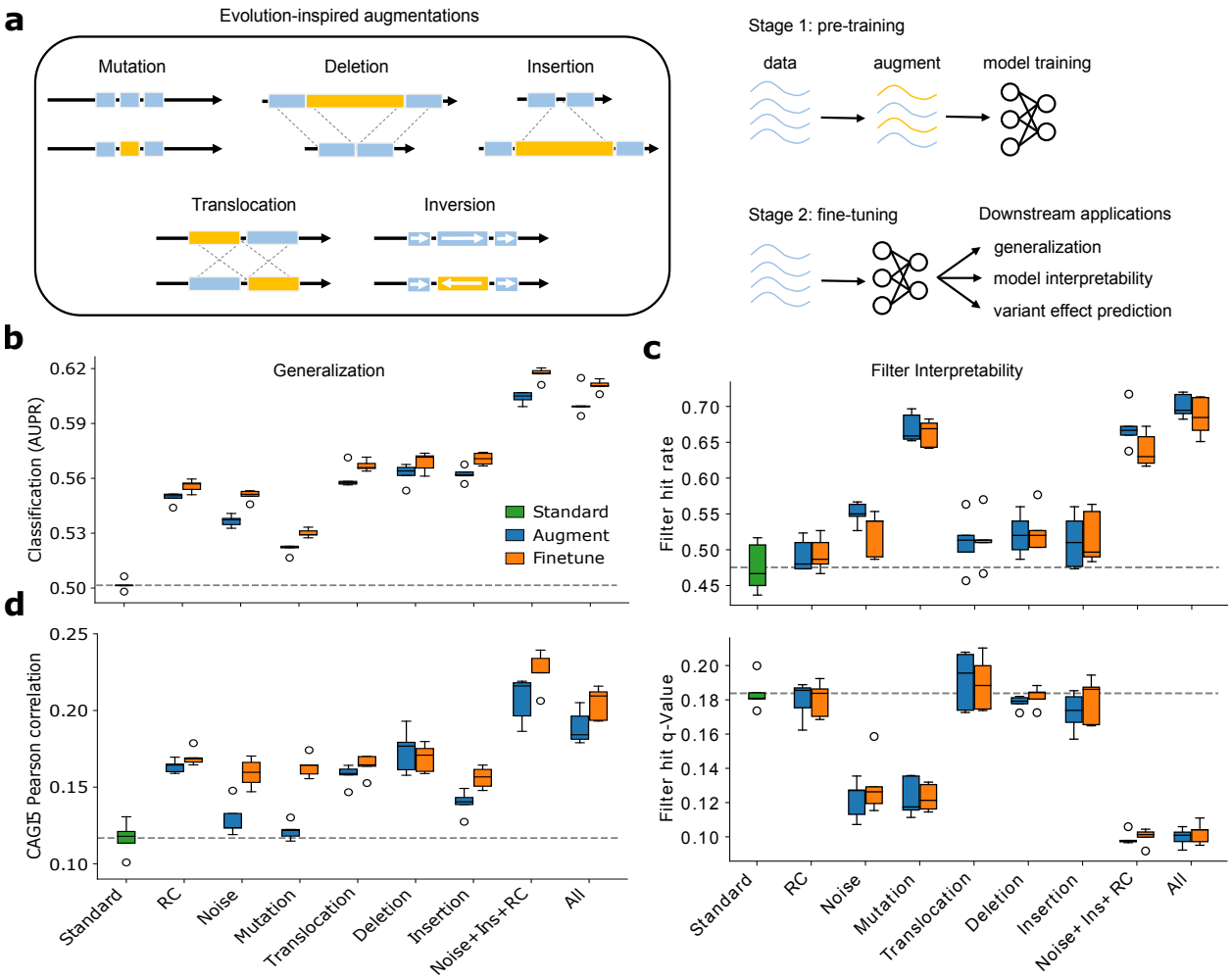


Figure 1. EvoAug improves performance. (**a**) Schematic of evolution-inspired data augmentations (left) and the two-stage training procedure (right). (**b**) Generalization performance (area-under the precision-recall curve) for Basset models pretrained with individual and combination of augmentations (i.e. Gaussian noise + insertion + reverse-complement; all augmentations) and fine-tuned on Basset dataset. Standard represents no augmentations during training. (**c**) Comparison of the average hit-rate of first layer filters to known motifs in the JASPAR database (top) and the average q-value of the filters with matches (bottom). (**d**) Comparison of the average Pearson correlation between model predictions and experimental data from CAGI5 Challenge. (**b**-**d**) Each box-plot represents 5 trials with random initializations.

To demonstrate the utility of EvoAug, we analyzed several established DNNs across three prominent types of regulatory genomic prediction tasks that span a range of complexity. First, we explored the Basset dataset[20], which consists of a multi-task binary classification of chromatin accessibility sites across 161 cell types/tissues. We retrained a Basset model with each augmentation applied independently and in two sets of combinations. Strikingly, EvoAug-trained DNNs yielded improved test performance across the board compared to no augmentations, with larger performance gains when augmentations were employed in combinations (Fig. 1b). In all cases, fine-tuning on the original data improved performance further, even when augmentation hyperparameters were grossly misspecified (Supplementary Fig. 1). From hyperparameter sweeps, we observed that inversions improved with larger sizes up to the sequence length, which is essentially a reverse-complement transformation (Supplementary Figs. 1, 3, and 4). Hence, inversions were excluded to reduce redundancy.

Interestingly, certain EvoAug augmentations, such as random mutations and combinations of data augmentations, can greatly improve motif representations learned by first layer convolutional filters (Fig. 1c). That is, the convolutional filters capture a larger repertoire of motifs and their representations better reflect known motifs, quantitatively and qualitatively.

An important downstream application of genomic DNNs is scoring the functional consequences of mutations. Following previous procedures[2,17], we compared model predictions with saturation mutagenesis of 15 *cis*-regulatory elements measured

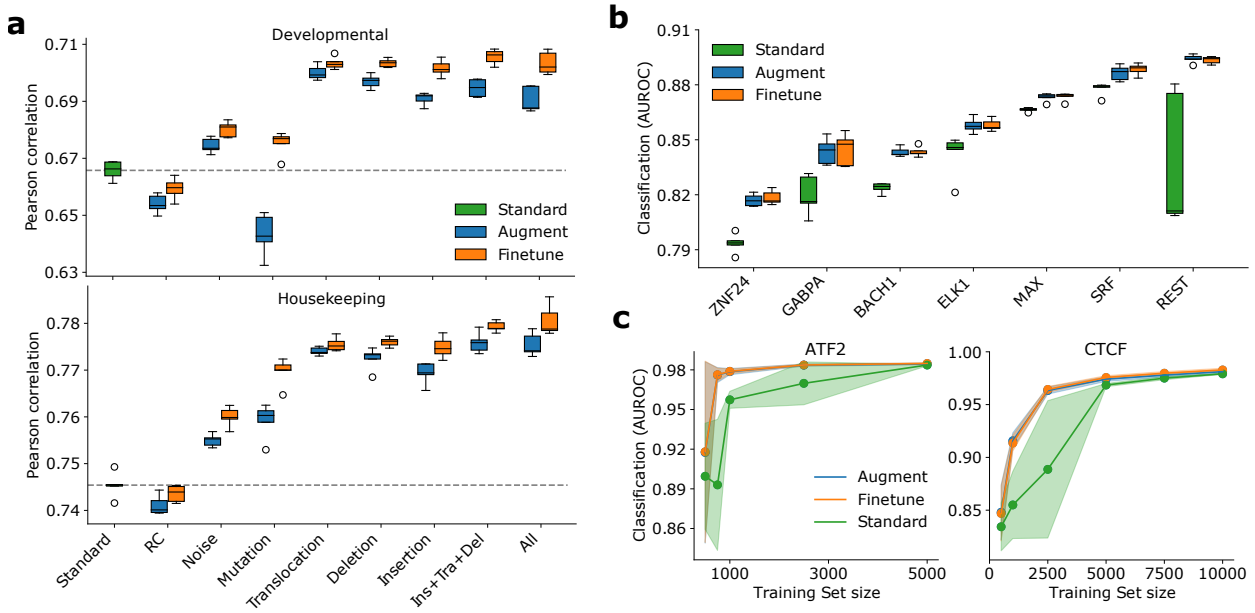


Figure 2. Generalization of EvoAug. (**a**) Box-plot of regression performance for DeepSTARR models pretrained with individual or combination of augmentations (i.e. insertion + translocation + deletion; all augmentations) and fine-tuned on original STARR-seq data for two promoters: developmental (top) and housekeeping (bottom). Standard represents no augmentations during training. (**b**) Box-plot of classification performance (area-under the receiver-operating-characteristic curve) for DNNs trained on ChIP-seq datas. **(c)** Average classification performance for ChIP-seq experiments downsampled to different dataset sizes. Shaded region represents the standard deviation of the mean. (**a**-**b**) Each box-plot represents 5 trials with random initializations.

experimentally with a massively parallel reporter assay – data collected through the CAGI5 Challenge[21]. As expected, EvoAug-trained DNNs outperformed standard training on this out-of-distribution generalization task (Fig. 1d). We note that pretrained DNNs with Gaussian noise and random mutagenesis augmentations performed worst for this task. This is because variant effects are local perturbations, but these augmentations impose flatness locally in sequence-function space. Notwithstanding, fine-tuning improves variant effect predictions beyond what was achieved with standard training.

To further demonstrate the effectiveness of EvoAug, we trained DeepSTARR models as a multi-task quantitative regression on STARR-seq data, which measures enhancer activity in *D. melanogaster* S2 cells. Similarly, the performance improved for most EvoAug augmentations, with the exception of reverse-complement and random mutations (Fig. 2a, Supplementary Figs. 3-5). As before, combinations of augmentations provided additional performance gains. Interestingly, attribution maps generated by EvoAug-trained models are qualitatively more interpretable with identifiable motifs and less spurious noise (Supplementary Fig. 6).

We also observed a systematic improvement in the generalization of EvoAug-trained DNNs across various single-task binary classification of TF binding from ChIP-seq data (Fig. 2b). However, fine-tuning did not improve performance, suggesting that the implicit prior imposed by EvoAug augmentations was suitable; the underlying regulatory grammars are not complex for these TFs. By retraining each DNN on down-sampled versions of two abundant ChIP-seq datasets, we found that EvoAug-trained DNNs improve performance by a larger margin for smaller datasets (Fig. 2c).

EvoAug greatly expands the set of data augmentations available for genomic DNNs. We found that EvoAug's two-stage training procedure of pretraining with augmentations followed by fine-tuning on the original data is effective, leading to improved generalization and model interpretability. This agrees with prior justifications for using evolution as a natural source for data augmentation[22]. Interestingly, the impact of evolutionary perturbations was not overwhelmingly disruptive, with enhanced performance even prior to fine-tuning. This functional robustness seems to be a feature of the non-coding genome[23]. EvoAug is readily accessible as an open-source PyTorch package and it should be broadly beneficial to enhance any sequence-based DNN in regulatory genomics.

Methods

Models and Datasets

Basset. The Basset dataset[20] consists of a multi-task binary classification of chromatin accessibility sites across 161 cell types/tissues. The inputs are genomic sequences of length 600 nt and the output are binary labels (representing accessible or not accessible) for 161 cell types measured experimentally using DNase-seq. We filtered sequences that contained at least one N character and the data splits (training, validation, test) reduced from (1,879,982; 70,000; 71,886) to (437,478; 16,410; 16,703). This "cleaned" dataset was analyzed using a Basset-inspired model, which is given according to:

- Input $x \in \{0,1\}^{600\times 4}$ (one-hot encoding of 600 nt sequence)
- 1D convolution (300 filters, size 19, stride 1)
- BatchNorm + ReLU
- Max-pooling (size 3, stride 3)
- 1D convolution (200 filters, size 11, stride 1)
- BatchNorm + ReLU
- Max-pooling (size 4, stride 4)
- 1D convolution (200 filters, size 7, stride 1)
- BatchNorm + ReLU
- Max-pooling (size 2, stride 2)
- Fully-connected (1000 units)
- BatchNorm + ReLU
- Dropout (0.3)
- Fully-connected (1000 units)
- BatchNorm + ReLU
- Dropout (0.3)
- Fully-connected output (161 units, sigmoid)

BatchNorm represents batch normalization[24], and dropout[25] rates set the probability that neurons in a given layer are temporarily removed during each mini-batch of training.

DeepSTARR. The DeepSTARR dataset[9] consists of a multi-task regression of enhancer activity for two promoters, well-known developmental and housekeeping transcriptional programs in *D. melanogaster* S2 cells. The inputs are genomic sequences of length 249 nt and the output is 2 scalar values representing the activity of developmental enhancers and housekeeping enhancers measured experimentally using STARR-seq. N characters were also removed, but overall, this minimally affected the size of the dataset. This dataset was analyzed using the original DeepSTARR model, given according to:

- Input $x \in \{0,1\}^{249\times 4}$
- 1D convolution (256 filters, size 7, stride 1)
- BatchNorm + ReLU
- Max-pooling (size 2, stride 2)
- 1D convolution (60 filters, size 3, stride 1)
- BatchNorm + ReLU
- Max-pooling (size 2, stride 2)
- 1D convolution (60 filters, size 5, stride 1)
- BatchNorm + ReLU
- Max-pooling (size 2, stride 2)
- 1D convolution (120 filters, size 3, stride 1)
- BatchNorm + ReLU
- Max-pooling (size 2, stride 2)
- Fully-connected (256 units)
- BatchNorm + ReLU
- Dropout (0.4)
- Fully-connected (256 units)
- BatchNorm + ReLU
- Dropout (0.4)
- Fully-connected output (2 units, linear)

ChIP-seq. Transcription factor (TF) chromatin immunoprecipitation sequencing (ChIP-seq) data was processed and framed as a binary classification task. The inputs are genomic sequences of length 200 nt and the output is a single binary label representing TF binding activity, with positive-label sequences indicating the presence of a ChIP-seq peak and negative-label sequences indicating it is a peak for a DNase I hypersensitive site from the same cell type but that does not overlap with any ChIP-seq peaks. Nine representative TF ChIP-seq experiments in a GM12878 cell line and a DNase-seq experiment for the same cell line were downloaded from ENCODE[26]; for details, see Supplementary Table 1. Negative sequences (i.e. DNase-seq peaks that do not overlap with any positive peaks) were randomly down-sampled to match the number of positive sequences, keeping the classes balanced. The dataset was split randomly into training, validation, and test set according to the fraction 0.7, 0.1, and 0.2, respectively.

A custom convolutional neural network was employed to analyze these datasets, given according to:

- Input $x \in \{0,1\}^{200\times 4}$
- 1D convolution (64 filters, size 7, stride 1)
- BatchNorm + ReLU
- Dropout (0.2)
- Max-pooling (size 4, stride 4)
- 1D convolution (96 filters, size 5, stride 1)
- BatchNorm + ReLU
- Dropout (0.2)
- Max-pooling (size 4, stride 4)
- 1D convolution (128 filters, size 5, stride 1)
- BatchNorm + ReLU
- Dropout (0.2)
- Max-pooling (size 2, stride 2)
- Fully-connected layer (256 units)
- BatchNorm + ReLU
- Dropout (0.5)
- Fully-connected output layer (1 unit, sigmoid)

Evolution-inspired Data Augmentations

EvoAug is comprised of a set of data augmentations given by the following:

- Mutation: a transformation where single nucleotide mutations are randomly applied to a given wild type sequence. This is implemented as follows: (1) given the hyperparameter of the fraction of nucleotides in each sequence to mutate (`mutate_frac`), the number of mutations for a given sequence length is calculated; (2) a position along the sequence is randomly sampled (with replacement) for each number of mutations; and (3) the selected positions are mutagenized to a random nucleotide. Since our implementation does not guarantee that a nucleotide selected will be mutated to a different nucleotide than it originally was, we account for silent mutations by dividing the user-defined `mutate_frac` by 0.75 so that on average the fraction of nucleotides in each sequence mutated to a different nucleotide is equal to `mutate_frac`.

- Translocation: a transformation that randomly selects a break point in the sequence (thereby creating two segments) and then swaps the order of the two sequence segments. An equivalent statement of this transformation is a "roll"—shifting the sequence forward along its length a randomly specified distance and then reintroducing the part of the sequence shifted beyond the last position back at the first position. This is implemented as follows: (1) given the hyperparameters of the minimum distance (`shift_min`, default 0) and maximum distance (`shift_max`) of the shift, the integer-valued shift length is chosen randomly from the interval $[-\texttt{shift_max}, -\texttt{shift_min}] \cup [\texttt{shift_min}, \texttt{shift_max}]$, where a negative value simply denotes a backward shift rather than a forward shift; and (2) the shift is applied to the sequence with a `roll` function in PyTorch.

- Insertion: a transformation where a random DNA sequence (of random length) is inserted randomly into a wild type sequence. This is implemented as follows: (1) given the hyperparameters of the minimum length (`insert_min`, default 0) and maximum length (`insert_max`) of the insertion, the integer-valued insertion length is chosen randomly from the interval between `insert_min` and `insert_max` (inclusive); and (2) the insertion is inserted at a random position within the original sequence. Importantly, to maintain a constant input sequence length to the model (i.e., original length plus `insert_max`), the remaining amount of length between the insertion length and `insert_max` is split evenly and placed on the 5' and 3' flanks of the wildtype sequence, with the remainder from odd lengths going to the 3' end. Whenever an insertion augmentation is employed in combination with other augmentations, all sequences without an insertion are padded with a stretch of random DNA of length `insert_max` at the 3' end to ensure that the model processes sequences with a constant length for both training and inference time.

- Deletion: a transformation where a random, contiguous segment of a wild type sequence is removed, and the shortened sequence is then padded with random DNA sequence to maintain the same length as wild type. This is implemented as follows: (1) given the hyperparameters of the minimum length (`delete_min`, default 0) and maximum length (`delete_max`) of the deletion, the integer-valued deletion length is chosen randomly from the interval between `delete_min` and `delete_max` (inclusive); (2) the starting position of the deletion is chosen randomly from the valid positions in the sequence that can encapsulate the deletion; (3) the deletion is performed on the designated stretch of the sequence; (4) the remaining portions of the sequence are concatenated together; and (5) random DNA is used to pad the 5' and 3' flanks to maintain a constant input sequence length, similar to the procedure for insertions.
- Inversion: a transformation where a random segment is replaced by its reverse-complement. This is implemented as follows: (1) given the hyperparameters of the minimum length (`invert_min`, default 0) and maximum length (`invert_max`) of the inversion, the integer-valued inversion length is chosen randomly from the interval between `invert_min` and `invert_max` (inclusive); (2) the starting position of the inversion is chosen randomly from the valid position indices in the sequence; and (3) the inversion (i.e., a reverse-complement transformation) is performed on the designated subsequence while the remaining portions of the sequence remain untouched.
- Reverse-complement: a transformation where the full sequence is replaced by its reverse-complement.
- Gaussian noise: a transformation where Gaussian noise (with properties `noise_mean`=0, and `noise_std`) is added to the input sequence.

Pretraining with data augmentations. Training with augmentations requires two main hyperparameters: first, a set of augmentations to sample from; and second, the maximum number of augmentations to be applied to a sequence. For each mini-batch during training, each sequence is randomly augmented independently. The number of augmentations to be applied to a given sequence has two settings in EvoAug: (1) `hard`, always equal to the maximum number of augmentations or (2) `soft` randomly select the number of augmentations for a sequence from 1 to the maximum number. Our experiments with Basset and DeepSTARR use the former setting, while our experiments with ChIP-seq datasets use the latter setting. Then, the subset of augmentations to be applied to the sequence is sampled randomly without replacement from the user-defined set of augmentations. After a subset of augmentations is chosen, the order in which multiple augmentations are applied to a single sequence is given by the following priority: inversion, deletion, translocation, insertion, reverse-complement, mutation, noise addition. Each augmentation is then applied stochastically for each sequence.

For the Basset and DeepSTARR models, each augmentation has an optimal setting that was determined from a hyperparameter search independently using the validation set (Supplementary Figs. 1, 3, and 4). For the Basset models, the hyperparameters were set to:

- mutation: `mutate_frac` = 0.15
- translocation: `shift_min` = 0, `shift_max` = 30
- insertion: `insert_min` = 0, `insert_max` = 30
- deletion: `delete_min` = 0, `delete_max` = 30
- reverse-complement: `rc_prob` = 0.5
- noise: `noise_mean` = 0, `noise_std` (standard deviation) = 0.3

For the DeepSTARR models, the hyperparameters were set to:

- mutation: `mutate_frac` = 0.05
- translocation: `shift_min` = 0, `shift_max` = 20
- insertion: `insert_min` = 0, `insert_max` = 20
- deletion: `delete_min` = 0, `delete_max` = 30
- reverse-complement: `rc_prob` = 0
- noise: `noise_mean` = 0, `noise_std` = 0.3

The same hyperparameter settings used by DeepSTARR all augmentations were used for the ChIP-seq analysis. For models trained with combinations of augmentations, the hyperparameters intrinsic to augmentations were set at the values identified above and the maximum number of augmentations per sequence was also determined through a hyperparameter sweep for each dataset (Supplementary Figs. 2 and 5).

Unless otherwise specified, all models were trained (with or without data augmentations) for 100 epochs using the Adam optimizer[27] with an initial learning rate of 1×10^{-3} and a weight decay (L_2 penalty) term of 1×10^{-6}; additionally, we employed early stopping with a patience of 10 epochs and a learning rate decay that decreased the learning rate by a factor of 0.1 when the validation loss did not improve for 5 epochs. For each model trained, the version of the model with the highest-performing weights during its training, as measured by validation loss, is the version of the model whose performance is reported here.

Fine-tuning. Models that completed training with data augmentations were subsequently fine-tuned on the original dataset without augmentations. Fine-tuning employs Adam optimizer with a learning rate of 1×10^{-4} and a weight decay (L_2 penalty) term of 1×10^{-6} for 5 epochs. The model that yields the lowest validation loss was used for test time evaluation.

Evaluation. When evaluating models on validation or test sets, no data augmentations were used on input sequences. For models trained with an insertion augmentation (alone or in combination with other augmentations), each sequence is padded at the 3' end with a stretch of random DNA of length `insert_max`.

Interpretability Analysis

Filter interpretability. We visualized the first-layer filters of various Basset models according to activation-based alignments[28] and compared how well they match motifs in the 2022 JASPAR nonredundant vertebrates database[29] using Tomtom[30], a motif search comparison tool. Matrix profiles MA1929.1 and MA0615.1 were excluded from filter matching to remove poor quality hits; low information content filters then to have a high hit rate with these two matrix profiles. Hit rate is calculated by measuring how many filters matched to at least one JASPAR motif. Average q-value is calculated by taking the average of the smallest q-values for each filter among its matches.

Attribution analysis. SHAP-based[31] attribution maps (implemented with `GradientShap` from the Captum package[32]) was used to generate sequence logos (visualized by Logomaker[33]) for sequences that exhibited high experimental enhancer activity for the Developmental promoter (i.e. task 0 in the DeepSTARR dataset). 1,000 random DNA sequences were synthesized to serve as references for each GradientShap-based attribution map. A gradient correction[34] was applied to each attribution map. For comparison, this analysis was repeated for a DeepSTARR model that was trained without any augmentations and a fine-tuned DeepSTARR model that was pretrained with all augmentations (excluding inversions) with two augmentations per sequence.

CAGI5 Challenge Analysis

The CAGI5 challenge dataset[21] was used to benchmark model performance on variant effect predictions. This dataset contains massively parallel reporter assays (MPRAs) that measure the effect size of single-nucleotide variants through saturation mutagenesis of 15 different regulatory elements ranging from 187 nt to 600 nt in length. We extracted 600 nt sequences from the reference genome centered on each regulatory region of interest and converted it into a one-hot representation. Alternative alleles are then substituted correspondingly to construct the CAGI test sequences.

For a given Basset model, the output predictions of two input sequences, one with a centered reference allele and the other with an alternative allele, are made. The cell type-agnostic approach employed in this study uses the mean across these values to calculate a single scalar value – functional activity across cell types. The effect size is then calculated with the log-ratio of this single value for the alternative allele and reference allele, according to: log(alternative value / reference value).

To evaluate the variant effect prediction performance, Pearson correlation was calculated within each CAGI5 experiment between the experimentally measured and predicted effect sizes. The average of the Pearson correlation across all 15 experiments represents the overall performance of the model.

Code Availability

EvoAug Python package is deposited on the Python Package Index (PyPI) repository with documentation hosted on https://evoaug.readthedocs.io. The open-source project repository is available at GitHub, https://github.com/p-koo/evoaug. The code to perform the analysis in this paper is available on GitHub, https://github.com/p-koo/evoaug_analysis.

Data Availability

Data and model weights are available at Zenodo: doi.org/10.5281/zenodo.7265991 (data) and doi.org/10.5281/zenodo.7277777 (weights).

References

1. Chen, K. M., Wong, A. K., Troyanskaya, O. G. & Zhou, J. A sequence-based global map of regulatory activity for deciphering human genetics. *Nat. Genet.* 1–10 (2022).
2. Avsec, Z. *et al.* Effective gene expression prediction from sequence by integrating long-range interactions. *bioRxiv* (2021).
3. Zhou, J. Sequence-based modeling of three-dimensional genome architecture from kilobase to chromosome scale. *Nat. Genet.* **54**, 725–734 (2022).

4. Hoffman, G. E., Bendl, J., Girdhar, K., Schadt, E. E. & Roussos, P. Functional interpretation of genetic variants using deep learning predicts impact on chromatin accessibility and histone modification. *Nucleic Acids Res.* **47**, 10597–10611 (2019).
5. Dey, K. K. *et al.* Evaluating the informativeness of deep learning annotations for human complex diseases. *Nat. Commun.* **11**, 1–9 (2020).
6. Koo, P. K. & Ploenzke, M. Improving representations of genomic sequence motifs in convolutional networks with exponential activations. *Nat. Mach. Intell.* **3**, 258–266 (2021).
7. Avsec, Ž. *et al.* Base-resolution models of transcription-factor binding reveal soft motif syntax. *Nat. Genet.* **53**, 354–366 (2021).
8. Koo, P. K., Majdandzic, A., Ploenzke, M., Anand, P. & Paul, S. B. Global importance analysis: An interpretability method to quantify importance of genomic features in deep neural networks. *PLoS Comput. Biol.* **17**, e1008925 (2021).
9. de Almeida, B. P., Reiter, F., Pagani, M. & Stark, A. Deepstarr predicts enhancer activity from dna sequence and enables the de novo design of synthetic enhancers. *Nat. Genet.* **54**, 613–624 (2022).
10. Horton, C. A. *et al.* Short tandem repeats recruit transcription factors to tune eukaryotic gene expression. *Biophys. J.* **121**, 287a–288a (2022).
11. Shorten, C. & Khoshgoftaar, T. M. A survey on image data augmentation for deep learning. *J. Big Data* **6**, 1–48 (2019).
12. Fort, S., Brock, A., Pascanu, R., De, S. & Smith, S. L. Drawing multiple augmentation samples per image during training efficiently decreases test error. *arXiv 2105.13343* (2021).
13. Zhu, S., An, B. & Huang, F. Understanding the generalization benefit of model invariance from a data perspective. *Adv. Neural Inf. Process. Syst.* **34**, 4328–4341 (2021).
14. Geiping, J. *et al.* How much data are augmentations worth? an investigation into scaling laws, invariance, and implicit regularization. *arXiv 2210.06441* (2022).
15. Puli, A., Zhang, L. H., Oermann, E. K. & Ranganath, R. Out-of-distribution generalization in the presence of nuisance-induced spurious correlations. *arXiv* (2021).
16. Zhou, H., Shrikumar, A. & Kundaje, A. Towards a better understanding of reverse-complement equivariance for deep learning models in genomics. *Mach. Learn. Comput. Biol.* 1–33 (2022).
17. Toneyan, S., Tang, Z. & Koo, P. K. Evaluating deep learning for predicting epigenomic profiles. *bioRxiv* (2022).
18. Kelley, D. R. Cross-species regulatory sequence activity prediction. *PLoS Comput. Biol.* **16**, e1008050 (2020).
19. Frazer, K. A., Murray, S. S., Schork, N. J. & Topol, E. J. Human genetic variation and its contribution to complex traits. *Nat. Rev. Genet.* **10**, 241–251 (2009).
20. Kelley, D. R., Snoek, J. & Rinn, J. L. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Res.* **26**, 990–999 (2016).
21. Shigaki, D. *et al.* Integration of multiple epigenomic marks improves prediction of variant impact in saturation mutagenesis reporter assay. *Hum. Mutat.* **40**, 1280–1291 (2019).
22. Lu, A. X., Lu, A. X. & Moses, A. Evolution is all you need: Phylogenetic augmentation for contrastive learning. *arXiv 2012.13475* (2020).
23. Kryukov, G. V., Schmidt, S. & Sunyaev, S. Small fitness effect of mutations in highly conserved non-coding regions. *Hum. Mol. Genet.* **14**, 2221–2229 (2005).
24. Ioffe, S. & Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Int. Conf. on Mach. Learn.* 448–456 (2015).
25. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The J. Mach. Learn. Res.* **15**, 1929–1958 (2014).
26. Luo, Y. *et al.* New developments on the encyclopedia of dna elements (encode) data portal. *Nucleic Acids Res.* **48**, D882–D889 (2020).
27. Kingma, D. & Ba, J. Adam: A method for stochastic optimization. *arXiv* 1412.6980 (2014).
28. Koo, P. K. & Ploenzke, M. Deep learning for inferring transcription factor binding sites. *Curr. Opin. Syst. Biol.* **19**, 16–23 (2020).
29. Castro-Mondragon, J. A. *et al.* JASPAR 2022: the 9th release of the open-access database of transcription factor binding profiles. *Nucleic Acids Res.* **50**, D165–D173 (2021).

30. Gupta, S., Stamatoyannopoulos, J. A., Bailey, T. L. & Noble, W. S. Quantifying similarity between motifs. *Genome Biol.* **8**, 1–9 (2007).

31. Lundberg, S. M. & Lee, S.-I. A unified approach to interpreting model predictions. *Adv. Neural Inf. Process. Syst.* **30** (2017).

32. Kokhlikyan, N. *et al.* Captum: A unified and generic model interpretability library for pytorch. *arXiv 2009.07896* (2020).

33. Tareen, A. & Kinney, J. B. Logomaker: beautiful sequence logos in python. *Bioinformatics* **36**, 2272–2274 (2020).

34. Majdandzic, A., Rajesh, C. & Koo, P. K. Statistical correction of input gradients for black box models trained with categorical input features. *bioRxiv* (2022).

Acknowledgements

This work was supported in part by funding from the NIH grant R01HG012131 and the Simons Center for Quantitative Biology at Cold Spring Harbor Laboratory. This work was performed with assistance from the US National Institutes of Health Grant S10OD028632-01. The authors would also like to thank Jakub Kaczmarzyk for help setting up the *Python Package Index* and code documentation on `ReadTheDocs.org`.

Author contributions statement

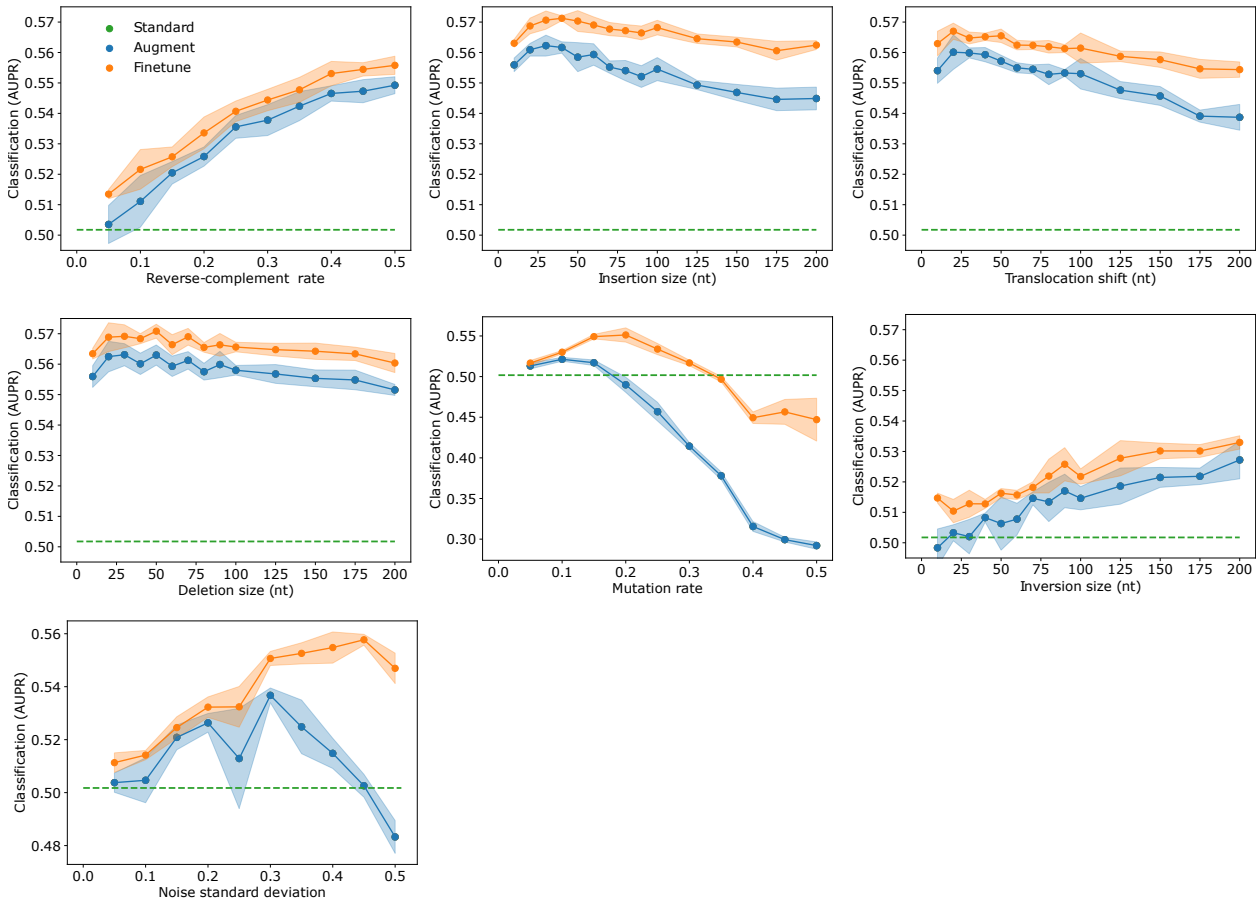
NKL and PKK conceived of the method, designed the experiments, and wrote the majority of the code base. NKL, ST, and ZT conducted experiments and analyzed the data. All authors interpreted the results and contributed to the manuscript.

Competing interests

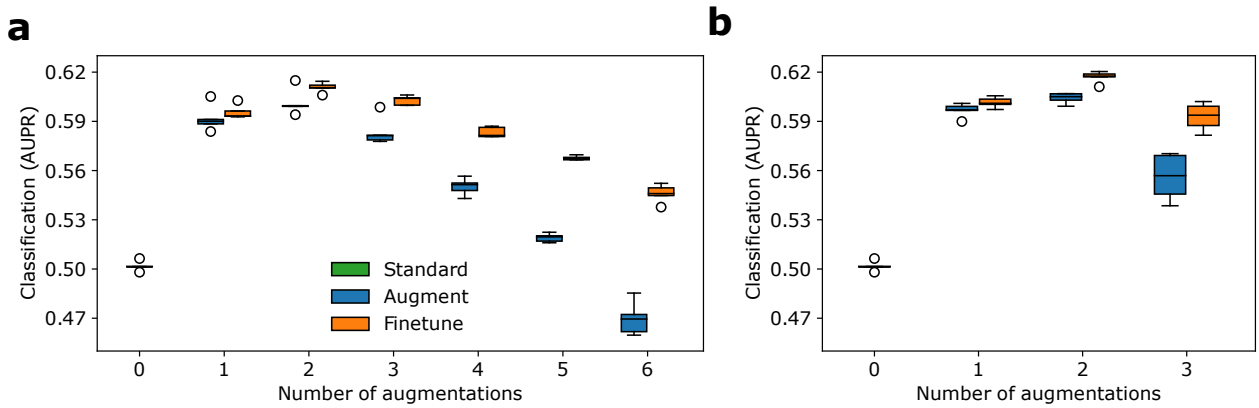
The authors declare no competing interests.

Supplementary Table 1. ENCODE ChIP-seq details. Ten representative TF ChIP-seq experiments in GM12878 cell line and a DNase-seq experiment (File accession: ENCFF235KUD) for the same cell line were downloaded from ENCODE. Table shows ENCODE file accession codes for all transcription factor proteins.

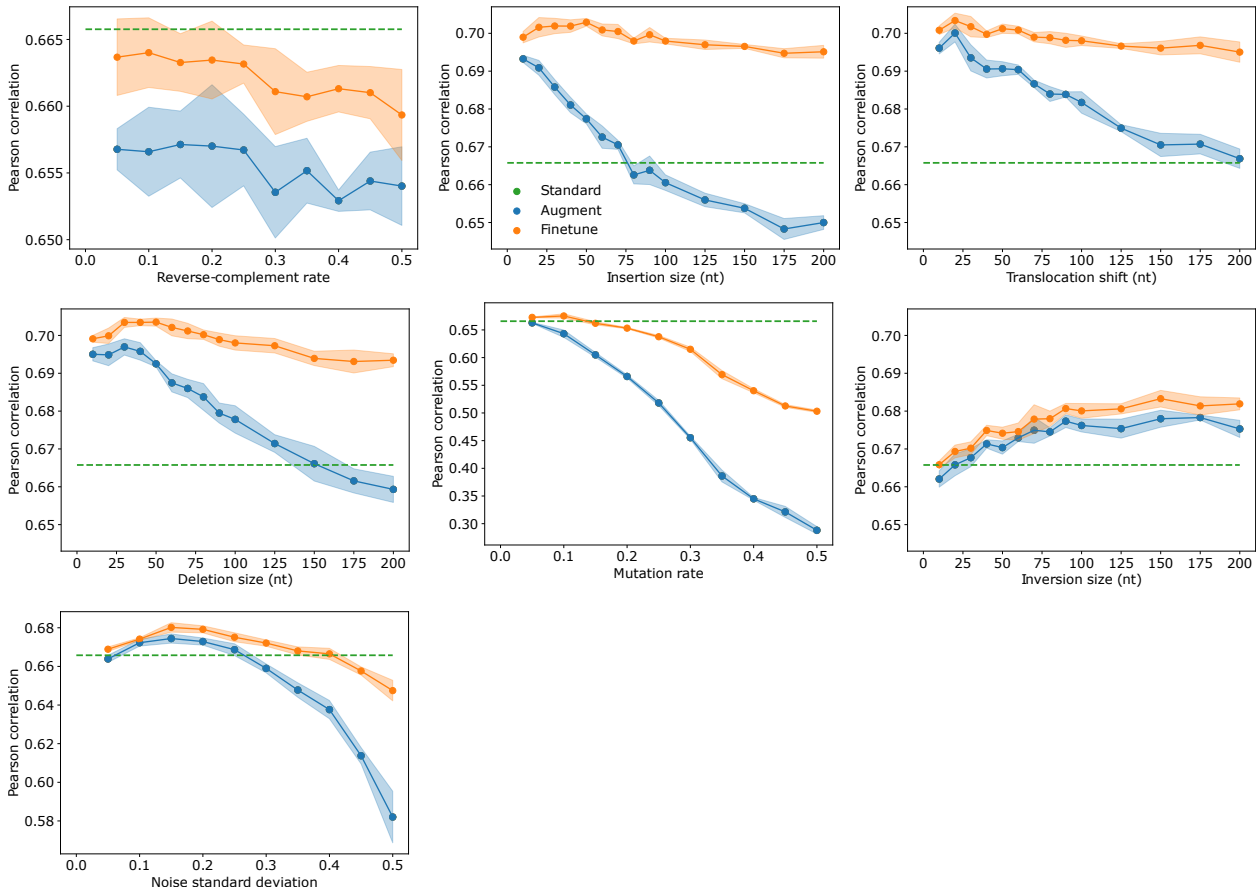
PROTEIN	ENCODE FILE ACCESSION	CELL LINE
MAX	ENCFF083KVY	GM12878
BACH1	ENCFF012JXJ	GM12878
GABPA	ENCFF116EXQ	GM12878
ZNF24	ENCFF103OOV	GM12878
ELK1	ENCFF556JBS	GM12878
SRF	ENCFF909FRA	GM12878
REST	ENCFF677KJB	GM12878
ATF2	ENCFF127GYQ	GM12878
CTCF	ENCFF710VEH	GM12878



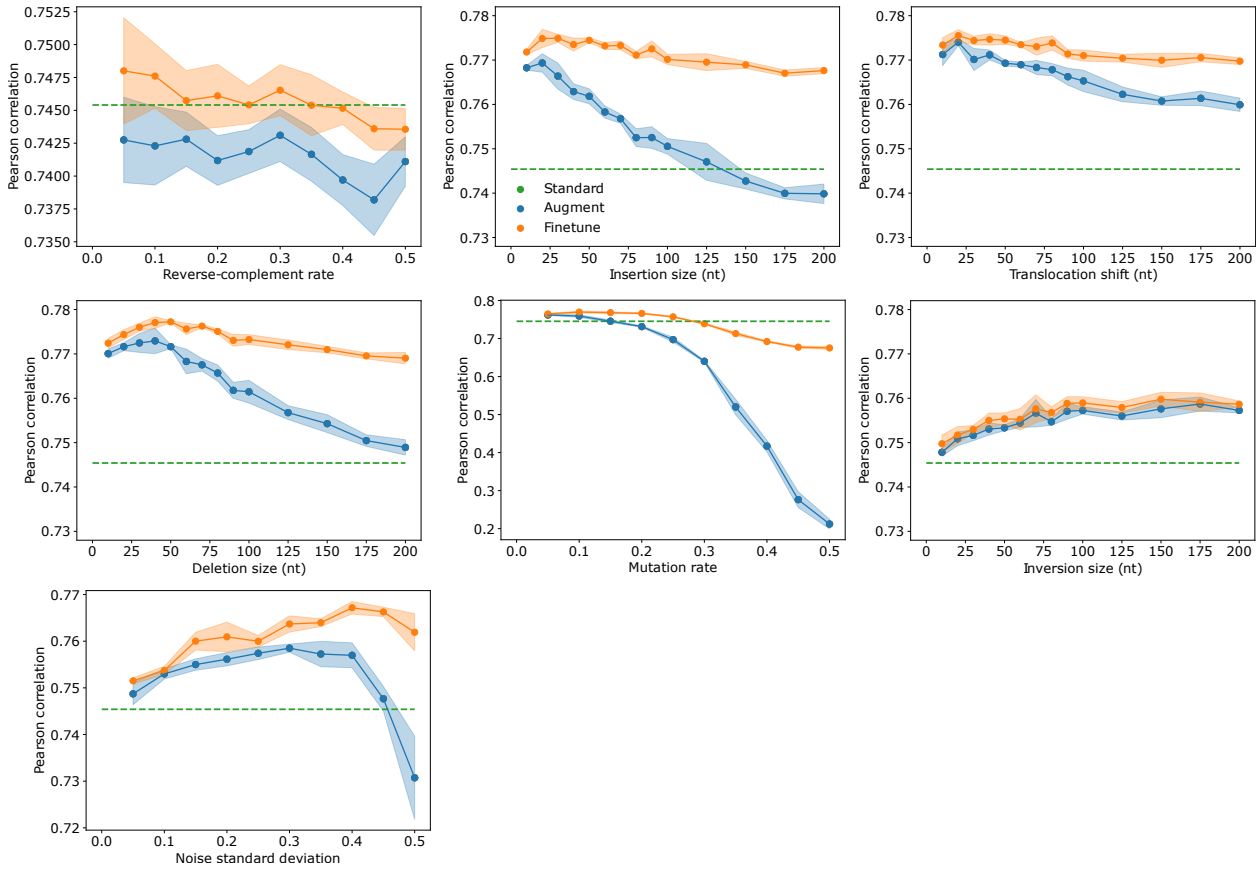
Supplementary Figure 1. Hyperparameter sweep of each augmentation method for Basset. Each plot shows the average classification performance for different hyperparameter values intrinisic to each data augmentation method. Shaded region represents the standard deviation of the mean. Dashed line represents the performance without augmentations. Values reported are with $n = 5$ trials with random initializations.



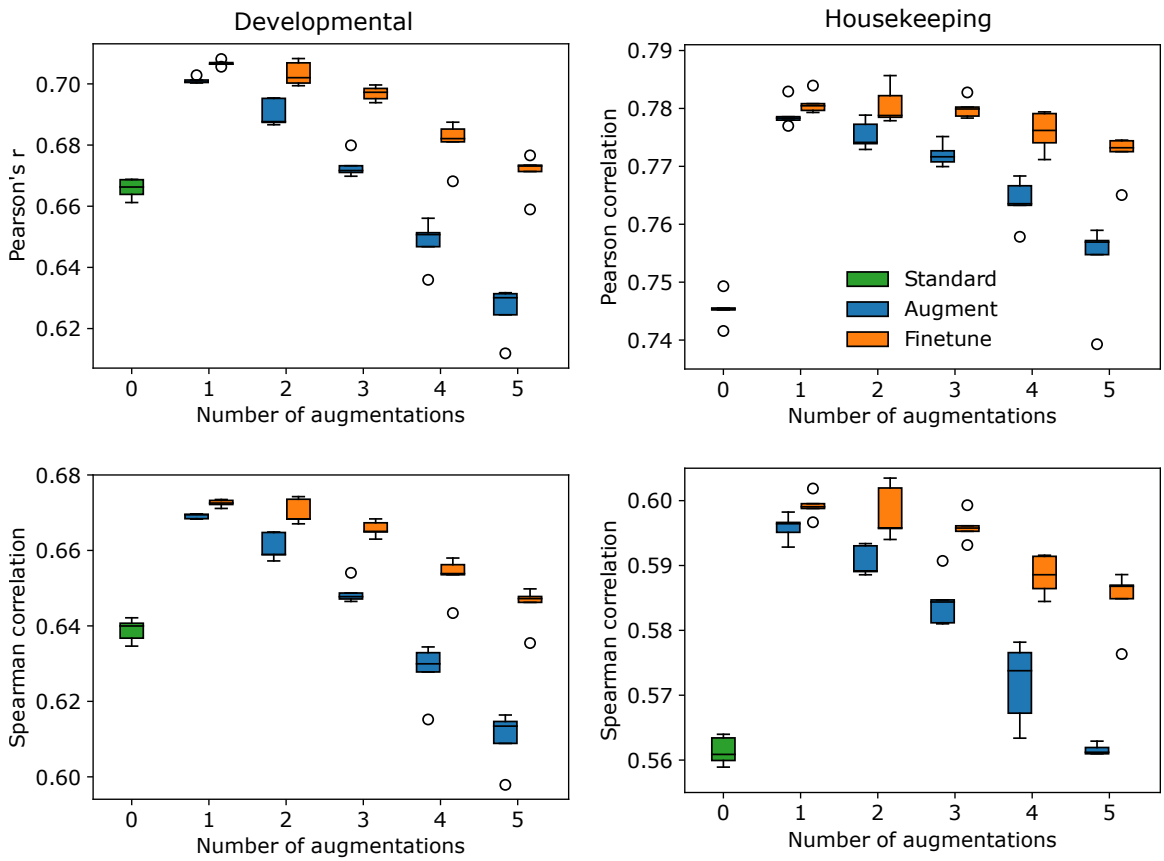
Supplementary Figure 2. Sweep in number of applied augmentations for Basset. (**a**) Box-plot of classification performance for Basset models pre-trained with two combination strategies (**a**) all and (**b**) insertion + translocation + deletion, and fine-tuned on original data. All represents reverse-complement, Gaussian noise, insertion, deletion, translocation, and mutation. Standard represents no augmentations during training. Each number in the x-axis represents the number of augmentations that are applied to each sequence in combinations during training. Box plots show the first and third quartiles, central line is the median, and the whiskers show the range of data with outliers removed. Values reported are with $n = 5$ trials with random initializations.



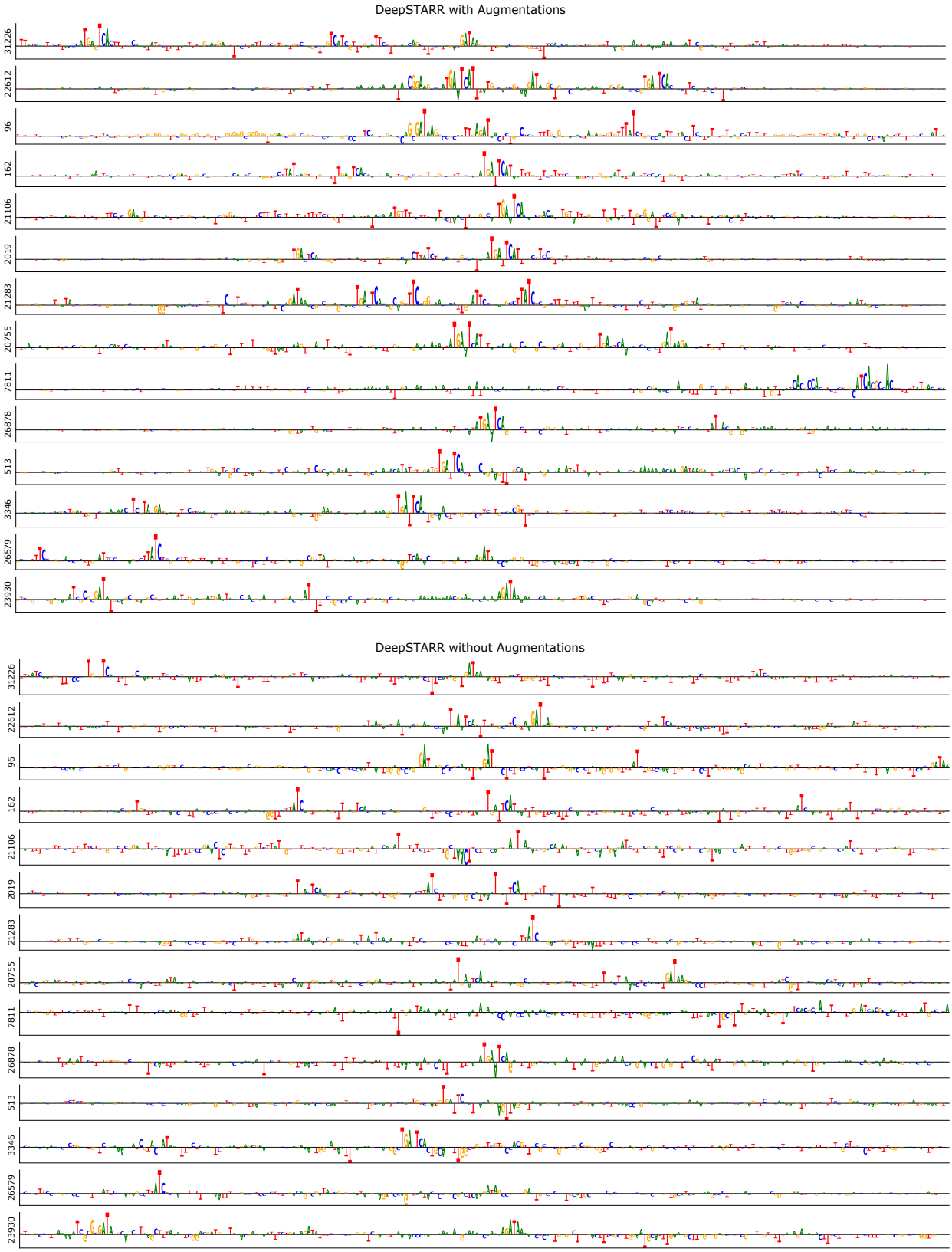
Supplementary Figure 3. Hyperparameter sweep of each augmentation method for DeepSTARR, Developmental. Each plot shows the average classification performance for different hyperparameter values intrinisic to each data augmentation method. Shaded region represents the standard deviation of the mean. Dashed line represents the performance without augmentations. Values reported are with $n = 5$ trials with random initializations.



Supplementary Figure 4. Hyperparameter sweep of each augmentation method for DeepSTARR, Housekeeping. Each plot shows the average classification performance for different hyperparameter values intrinisic to each data augmentation method. Shaded region represents the standard deviation of the mean. Dashed line represents the performance without augmentations. Values reported are with $n = 5$ trials with random initializations.



Supplementary Figure 5. Sweep in number of applied augmentations for DeepSTARR. Box-plot of regression performance (i.e. Pearson's r for top row and Spearman correlation for bottom row) for DeepSTARR models pre-trained with all augmentations–namely reverse-complement, Gaussian noise, insertion, deletion, translocation, and mutation–for developmental (left) and housekeeping (right) conditions. Standard represents no augmentations during training. Each number in the x-axis represents the number of augmentations that are applied to each sequence in combinations during training. Box plots show the first and third quartiles, central line is the median, and the whiskers show the range of data with outliers removed. Values reported are with $n = 5$ trials with random initializations.



Supplementary Figure 6. Attribution map comparison for DeepSTARR. Sequence logos of SHAP-based attribution maps for a fine-tuned DeepSTARR model that was pretrained with a combination of all augmentations up to 2 augmentations total per sequence (top) and DeepSTARR trained without any augmentations (bottom). The number on the *y*-axis represents the index of the sequence from the DeepSTARR test set. Sequence logos of GATA and AP1 are shown for visual comparison.