DREAM Challenge 2022
Predicting gene expression using millions of random promoter sequences
by
High Schoolers Are All You Need

Abstract

Deep learning has proven to be a powerful tool for prediction of regulatory genomic function. In particular, deep convolutional neural networks (CNNs) and transformers are exceptionally powerful due to their ability to learn the robust patterns that control the expression of promoter sequences; CNNs enable learning of motifs that control expression, and attention accounts for position dependent interactions between such motifs. However, many modern CNN advances in image processing and transformer studies in genomics remain unexplored when combined. Specifically, the convnext network's strategies, such as 1x1 convolutions, and the enformer's relative positional encoding offer promising approaches that can improve performance. Here, we repurpose convnext's 1x1 convolutions for the transformer layer, to preserve positional information before the fully connected layers at the end of the network. Moreover, instead of standard max pooling after convolutions, we employ attention pooling, however again modified to prevent feature information from crossing over positions. Strikingly, by combining these design modifications and CNN layers with residually connected dilated convolutions, we are able to achieve much higher performance than baseline CNNs. We find attention to be critical in the success of our model, demonstrating that modeling cooperativity between convolution-extracted features leads to improved performance. Together, this work demonstrates how combining modern CNN architectures with multi-head attention can lead to vastly improved performance for predicting genomic expression values.

1. Description of data usage

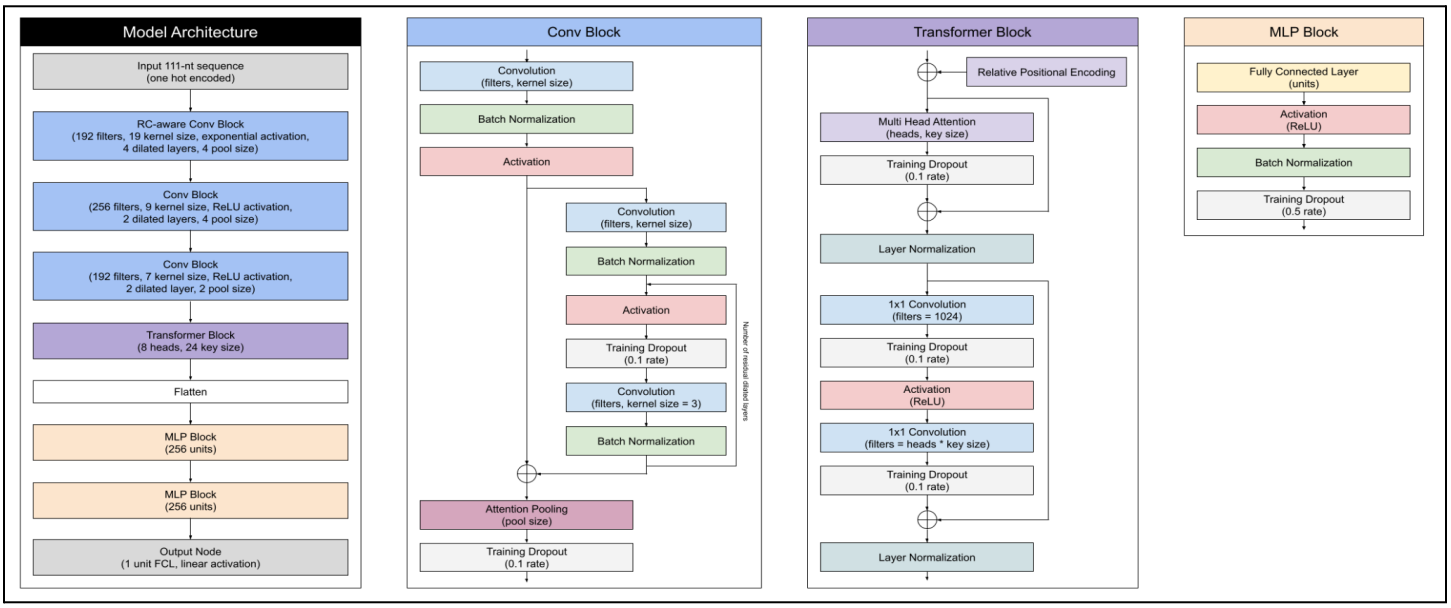
Using the provided testing and training sequence txt files, we generated a one-hot encoded dataset where each channel symbolizes a nucleotide (A, C, G, T, respectively); unknown nucleotides (i.e. Ns) were substituted with 0s in all four channels. We removed all sequences with lengths greater than 111-nts from our dataset, and padded all shorter sequences with 0s such that all sequences were of shape (111, 4). Of the processed training sequences, the last 104,448 were set aside for held-out validation. The first 6.6 million processed sequences were used for model training. Testing sequences were processed similar to training: one hot encoded and padded to length 111.

2. Description of the model

We employed a CNN supplemented with a transformer encoder as the baseline architecture for our highest performing model. Three convolutional blocks were used to extract import features from the input sequence, building latent representations in a hierarchical manner with exponentially growing receptive fields. Following the convolutions was a single, wide transformer layer to model interactions between extracted features. The final component of our model was two fully connected layers that analyze the identified features to produce a prediction.

The central component of each convolutional block is the convolutional layer which passes filters over the input. In the first convolutional block, a reverse complement aware convolution was used [1], which consists of two sets of tied weights; the second set of filters is equal to the first set, however the length and channel dimensions are reversed to simulate scanning a reverse complement sequence. Batch normalization was applied to the feature maps after each convolution, converting the data to z-scores with a subsequent rescaling [2]. A nonlinear activation function was implemented after normalization to allow the network to modulate information flow across layers; exponential activation was used in the first convolutional layer of the first convolutional block due to its highly divergent nature [3], while ReLU was used in all other layers. A series of dilated convolutions (each with progressively larger dilation rates by a factor of 2) were added in a residual connection to allow the network to learn corrections for representations in the main track via analysis of a

larger sequence context (while preserving resolution) [4]. See figure below for detailed diagram of the convolutional block.



A pooling mechanism was employed at the end of convolution blocks to constrict spatial information, thus increasing computational efficiency while reducing overfitting. However, as opposed to the standard max pooling widely used in deep learning, we designed a modified version of attention pooling presented in Enformer [5]. While max pooling is effective at preserving important information, defined by large values, it can have adversarial effects when grouping large pool sizes - context and absence information is often lost. Attention pooling offers a promising alternative, as it accomplishes the goal of forcing the model to learn generalizations by limiting information available for “memorization” without the flaws that accompany max pooling - through a learnable latent representation, the model can control how information is concatenated within pool windows. In this sense, overfitting is discouraged with less severe costs to the model’s expressivity. In the Enformer’s attention pooling, a latent representation is constructed by applying a fully connected layer to the input feature map; softmax is subsequently applied to generate a set of weights for each pooling window. These weights are then used to obtain a weighted average of positions in each pool window. However, by computing the latent representation in this manner, the model is given an abundance of parameters and feature information is able to cross over positions in between convolutional layers. This opens the door to overfitting in that cross positional information is unnecessary at this stage of the network, where the model should only be looking for patterns - interactions are modeled in the later transformer and fully connected layers. To amend this without drastically limiting the model’s expressivity we modified Enformer’s attention pooling with 1x1 convolutions (inspired by ConvNext [6]) instead of a fully connected layer to generate the latent representation; as such, feature information is contained within each position with the exception of minor adjacent/contextual interactions within each pool window.

After the three convolutional blocks, we added a transformer layer to capture interactions between features/motifs extracted by the convolutions [7]. The vanilla transformer was implemented with some modifications. The multi-head attention layer was supplemented with relative positional encodings according to Enformer [5]. The nonlinear transformation following the attention layer was manipulated to resemble an inverted bottleneck, where the features expand and then contract back to their original size; ReLU activation was applied between layers to induce nonlinearity. Interestingly, similar to attention pooling, we elevated our performance by modifying the transformer’s feed forward layers to analyze positions individually (with 1x1 convolutions) rather than collectively with fully-connected layers.

The final stage of our network consists of two fully connected layers, each followed by batch normalization and ReLU activation; the final output node (linear activation) represents the model’s prediction.

We interspersed dropout layers throughout the model as shown in the figure above to provide additional regularization [8]. During testing, the output predictions were converted to a uniform distribution by dividing the data into quantiles and scaling accordingly.

3. Training procedure

The settings for training and fine-tuning the model are listed in **Table 1**. We employed Adam [9] as the base optimizer with stochastic weight averaging (SWA) as the optimization procedure [10]. The epoch at which SWA was set to begin was 150 and the synchronization period was set to 30. The initial learning rate was set to 1e-4 during training and changed to 3e-5 after epoch 750 for fine-tuning. For each epoch in the designed training loop, the model is trained on a randomly chosen 307,200 sequence slice of the training data; model parameters are updated after every batch of 1024 sequences. Evaluation on the validation set is performed every three epochs with Pearson correlation coefficient (PCC) and Spearman's rank correlation coefficient, aligned with the competition's metrics. Additionally, the model was saved after every improvement in PCC. After fine-tuning the model, we chose the weights of the model saved at the peak of the validation Pearson correlation as the best model and used it to predict the expressions of the test sequences. The best performances of the model on the validation set in training and validation set are shown in **Table 2**.

Settings	**Training stage**	**Fine-tuning stage**
optimizer	Adam	Adam
optimization procedure	stochastic weight averaging	stochastic weight averaging
base learning rate	1e-4	3e-5
optimizer average period	30	30
optimizer momentum	β_1, β_2 = 0.9, 0.999	β_1, β_2 = 0.9, 0.999
batch size	1024	1024
slice size	307,200	307,200
training epochs	750	100
loss function	mean-squared error	mean-squared error

Table 1: Training and Fine-tuning settings for the highest performing model

	Pearson's r	**Spearman rho**
training	0.7469	0.763
fine-tuning	0.7474	0.764

Table 2: The best performances of the model

4. Other important features

See attention pooling description in Section 2.

5. Contributions and Acknowledgement

5.1 Contributions

Name	Affiliation	Email
Rohan Ghotra	Syosset High School, Cold Spring Harbor Laboratory	ghotra@cshl.edu
Yiyang Yu	Syosset High School, Cold Spring Harbor Laboratory	yiyu@cshl.edu
Ethan Labelson	Friends Academy, Cold Spring Harbor Laboratory	labelso@cshl.edu
Aayush Prakash	Half Hollow Hills High School	aayush.prakash@outlook.com
Ashwin Narayanan	Jericho High School	nashwin2005@gmail.com
Peter Koo	Simons Center for Quantitative Biology, Cold Spring Harbor Laboratory	koo@cshl.edu

5.2 Acknowledgement

We would like to thank Jakub Kaczmarzyk and Ziqi (Amber) Tang for help with model hyperpameter search using W&B and for discussions throughout the competition.

6. References

[1] Shrikumar, Avanti, Peyton Greenside, and Anshul Kundaje. "Reverse-complement parameter sharing improves deep learning models for genomics." *BioRxiv* (2017): 103663.

[2] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *International conference on machine learning*. PMLR, 2015.

[3] Koo, Peter K., and Matt Ploenzke. "Improving representations of genomic sequence motifs in convolutional networks with exponential activations." *Nature Machine Intelligence* 3.3 (2021): 258-266.

[4] Yu, Fisher, and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions." *arXiv preprint arXiv:1511.07122* (2015).

[5] Avsec, Žiga, et al. "Effective gene expression prediction from sequence by integrating long-range interactions." *Nature methods* 18.10 (2021): 1196-1203.

[6] Liu, Zhuang, et al. "A convnet for the 2020s." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.

[7] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

[8] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15.1 (2014): 1929-1958.

[9] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

[10] Izmailov, Pavel, et al. "Averaging weights leads to wider optima and better generalization." *arXiv preprint arXiv:1803.05407* (2018).

7. Feedback