DREAM Challenge 2022
Predicting gene expression using millions of random promoter sequences
by
Camformers

Abstract

We built a convolutional neural network (CNN) with residual connections to predict reporter gene expression in S. cerevisiae based on a 110 nucleotide (nt) DNA sequence representing a minimal promoter. The sequences consisted of an 80 nt variable region embedded between two fixed sequences of length 17 and 13 nt, respectively. The model included six convolutional layers with three residual connections allowing the model to bypass every other layer. Batch normalisation and dropouts were used for regularisation. A max pooling operation was added after the penultimate convolutional layer to reduce model size and improve generalisation. The output of the final convolutional layer was flattened into 13,312 features and fed into a block of two dense layers outputting 256 features, followed by a final dense layer outputting the predicted expression level as a float value. All layers except the last used a rectified linear unit activation. The whole model had 16,611,073 trainable parameters. Implementation was done in PyTorch and total runtime from raw data to predictions was ~16 hours using one core on a TPU v3-8. Model performance during training was r=0.748 (r2=0.560) and ρ=0.765 on our internal validation set (10%). Final performance on the challenge leaderboard (13% of external test data) was r=0.962 (r2=0.926), ρ=0.967, ScorePearsonR²=0.763, and ScoreSpearman=0.823.

Final submission: https://github.com/FredrikSvenssonUK/camformers_submission

1. Description of data usage

1.1. Data preprocessing

Training data (*n*=6,739,258 sequences) were downloaded from the challenge website. Only sequences with a length within 110 ±3 nucleotides (nt) were retained (110,138 dropped) and sequences containing more than three "N" were discarded (23,533). Sequences shorter than 110 nt were padded with "N" at the end, and sequences longer than 110 nt were truncated at position 110.

1.2 Data encoding

The preprocessed sequences (*n* = 6,605,587) were one-hot-encoded using A=(1,0,0,0), C=(0,1,0,0), G=(0,0,1,0), and T=(0,0,0,1). Target values for the sequences were used as supplied without any transformation. Test data (*n*=71,103 sequences) were downloaded from the challenge website and one-hot-encoded.

1.3 Training and validation sets

The encoded training data were randomly split into training (90%, *n*=5,945,028 sequences) and validation (10%, *n*=660,559 sequences) sets using the `train_test_split` function from scikit-learn.

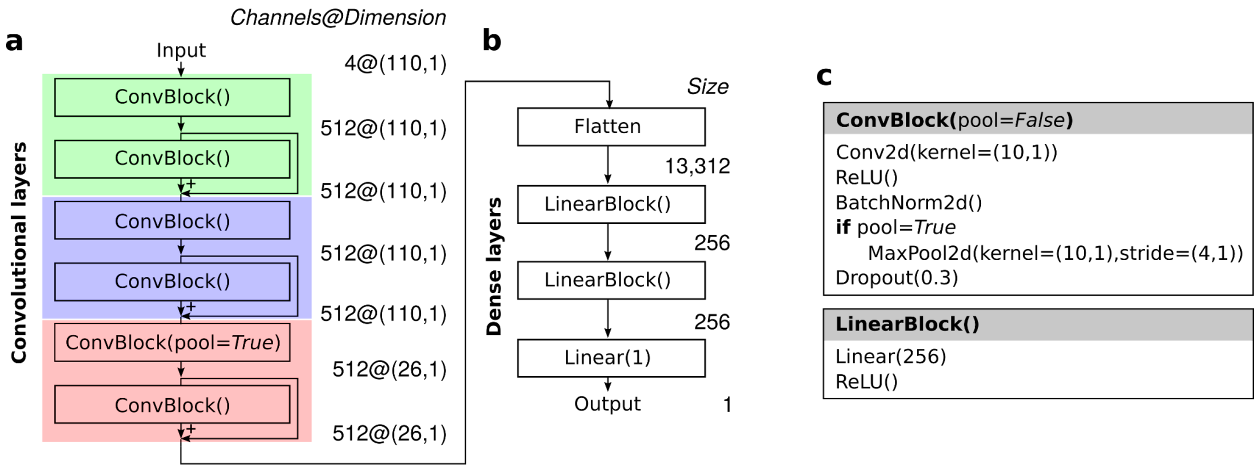

***Figure 1. Overview of model architecture.** Convolutional layers. Six convolutional layers with residual connections implemented over every other layer. **b** Dense layers. The output of the final convolutional layer is flattened and run through three dense layers. The final layer outputs the predicted expression. **c** Definition of custom blocks (ConvBlock and LinearBlock) used in the overview. ConvBlock takes a "pool" option (False by default), which determines whether a max pooling operation is performed.*

2. Description of the model

We used a convolutional neural network with residual connections (He et al. 2016) implemented in PyTorch (v1.11). The model had a total of 16,611,073 trainable parameters and was built using the following architecture (Figure 1):

Model input

One-hot-encoded DNA sequence(s). Input shape: (channels: 4, height: 110, width: 1).

Convolutional layers

- Convolution layer 1_0; kernel_size: (10, 1), in_channels: 4, out_channels: 512
- Convolution layer 1_1; kernel_size: (10, 1), in_channels: 512, out_channels: 512
- Convolution layer 2_0; kernel_size: (10, 1), in_channels: 512, out_channels: 512
- Convolution layer 2_1; kernel_size: (10, 1), in_channels: 512, out_channels: 512
- Convolution layer 3_0; kernel_size: (10, 1), in_channels: 512, out_channels: 512
- Convolution layer 3_1; kernel_size: (10, 1), in_channels: 512, out_channels: 512

The convolutional layers were connected sequentially, but output from layers 1_0, 2_0, and 3_0 was added to the output of layers 1_1, 2_1, and 3_1, respectively. A rectified linear unit (ReLU) activation, batch normalisation (BatchNorm2d), and dropout rate 0.3 were added after each convolutional layer. A max pooling (MaxPool2d) operation using kernel shape (10, 1) and stride (4, 1) was implemented after convolutional layer 3_0. Data size after the pooling operation was (26, 1).

Dense layers

- Dense layer 1; in_features: 26*512, out_features: 256

- Dense layer 2; in_features: 256, out_features: 256
- Dense layer 3; in_features: 256, out_features: 1

ReLU activations were added after dense layer 1 and 2.

3. Training procedure

Batch size 256 was used for training. The model was optimised using `AdamW` (lr=1e-3, weight_decay=1e-3) together with L1 loss (absolute error) using the `torch.nn.L1Loss` function. Moreover, `torch.optim.lr_scheduler.ReduceLROnPlateau` was used to modify the learning rate by monitoring the validation loss with a patience of 10.

Early stopping was implemented based on Pearson (r) and Spearman (ρ) correlation on the validation set and training stopped after 10 epochs of no improvement in r+ρ. These metrics were implemented based on the Python `stats.spearmanr` and `stats.pearsonr` functions. The model from the best epoch (r+ρ) was saved and used for final predictions. Best performance was observed after 24 epochs (Table 1, Figure 2).

Table 1. Performance metrics for best epoch.

Epoch	T loss	V loss	T r	V r	T ρ	V ρ
24	1.1200	1.1509	0.7581	0.7481	0.7748	0.7655

Abbreviations: T, training set; V, validation set; r, Pearson correlation, ρ, Spearman correlation.

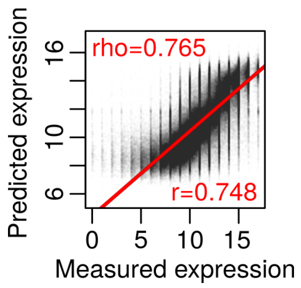

Figure 2. Model performance during training. *Measured versus predicted expression for 660,559 sequences in the validation set after 24 training epochs. The red line represents a linear regression fit with r value as indicated.*

Model training was performed on a TPU v3-8 tpu-vm-pt-1.11. Total time for end-to-end run (including data preprocessing, encoding, training, and prediction of the test set) was about 16 hours.

4. Other important features

4.1 Model design

To identify the best performing model, different architectures were created and explored by varying model parameters and hyperparameters using Optuna. Optuna was used to vary batch size, learning rate, weight decay, size of the convolutional kernels, number of convolutional and dense layers, number of channels, and dropout rates. Finally, the max pooling operation was added based on insights from optimisation of alternative models (CNN and transformer models) suggesting that a max pooling operation at the penultimate layer improved generalisation.

5. Contributions and Acknowledgement

5.1 Contributions

Contributors are listed alphabetically. For queries, please contact Susanne Bornelöv.

Name	Affiliation	Email
Susanne Bornelöv	University of Cambridge	susanne.bornelov@cruk.cam.ac.uk
Fredrik Svensson	University College London	f.svensson@ucl.ac.uk
Maria-Anna Trapotsi	University of Cambridge	marianna.trapoti@cruk.cam.ac.uk

5.2 Acknowledgement

We are grateful to Kamal Kishore and Rory Stark for valuable discussions and support in installing PyTorch and TensorFlow on the HPC cluster at the Cancer Research UK Cambridge Institute, University of Cambridge. We thank Google’s TPU Research Cloud (TRC) for access to cloud TPUs throughout the competition.

6. References

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. “Deep Residual Learning for Image Recognition.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2016.90.

7. Feedback

Thank you for organising this interesting and highly engaging competition. For future challenges, we would like to suggest that deadlines should be clearer and more specific, preferably with timezone indicated.