Optimizing Sequence Data Analysis Using Convolution Neural Network for the Prediction of CNV Bait Positions

Zoltán Maróti[1*†], Peter Juma Ochieng[2,3*†], József Dombi[3,4], Miklós Krész[5,6], Tibor Kalmár[1*]

[1]Albert Szent-Györgyi Health Centre, University of Szeged, Korányi fasor 14-15, Szeged, H-6725, Csongrád-Csanád, Hungary.
[2]Interdisciplinary Research Development and Innovation Center of Excellence, Institute of Informatics, University of Szeged, Árpád tér 2, Szeged, H-6720, Csongrád-Csanád, Hungary.
[3]HUN-REN SZTE Research Group on Artificial Intelligence, University of Szeged, Árpád tér 2, Szeged, H-6720, Csongrád-Csanád, Hungary.
[4]Institute of Informatics, University of Szeged, Árpád tér 2, Szeged, H-6720, Csongrád-Csanád, Hungary.
[5]InnoRenew CoE, Livade 6a, Izola, SI-6310, Slovenia.
[6]Department of Applied Informatics, University of Szeged, Boldogasszony sgt. 6, Szeged, H-6725, Hungary.

*Corresponding author(s). E-mail(s): maroti.zoltan@med.u-szeged.hu; juma@inf.u-szeged.hu; kalmar.tibor@med.u-szeged.hu;
Contributing authors: dombi@inf.u-szeged.hu; miklos.kresz@innorenew.eu;
†These authors contributed equally to this work.

Supplementary Material

Nucleotide Sequence Homology Test

To avoid possible batch effects and overtraining due to excessive amount of homologous sequences in the train data we performed a many-to-many nucleotide homology sequence analysis with the MMSeqs2 analysis framework [1].
The nucleotide sequences corresponding to the target regions were extracted from the hs37d5 reference genome by

```
bedtools getfasta --fi/ssd/refseqs/hs37d5/hs37d5.fa--bed
CHUNKS.bed >chunks.fa
```

The nucleotide database for Mmseq2 was built by

```
mmseqs createdb chunks.fa seqDB --dbtype 2 --shuffle 0
```

for the many-to-many sequence comparison and clustering.
The clustering was performed for minimum 500 base pair of nucleotides (our smaller data segment size) with a permissive 80% of homology threshold using the

```
mmseqs linclust seqDB linClus80 tmp --remove-tmp-files 1
--min-aln-len 500\
--min-seq-id 0.8 -c 0.7 --max-seq-len 100000 --kmer-per-seq
1000 --cov-mode 1\
--alignment-mode 3 --cluster-mode 2 --spaced-kmer-mode 0
--kmer-per-seq-scale 0\
--threads 40
```

command. The clusters were converted to ".tsv" format with the

```
mmseqs createtsv seqDB seqDB linClus80 linClus80.tsv
```

And lastly, we counted the unique representative clusters (according to the data format https://mmseqs.com/latest/userguide.pdf, the cluster TSV format has two columns: the first is the **#cluster-representative**, and the second is the ID of the cluster-member) with the following commands:

```
cut -f 1 linClus80.tsv | uniq | wc -l
```

Our analysis identified 96,609 unique clusters from the 99,188 nucleotide sequences. Consequently, only approximately 2.6% of the regions exhibited more than 80% homology within the WES target regions. A key consideration in the design of oligo baits for WES capture is to avoid targeting homologous sequences, as these can lead to excessive off-target captures, ultimately increasing per-sample sequencing costs. Therefore, despite the low fraction of homologous sequences within the target regions, we expect minimal overlap between the bait regions and these homologous sequences.

1D CNN Model Architecture

The proposed 1D CNN model incorporates the targeted region, sequence information, and experimental coverage as input. The model then combines convolutional layers, batch normalization, reverse weighting for an uneven number of different difficulty training examples (based on the number of potentially overlapping baits in the data segment), dropout regularization, max pooling, and dense layers to train and predict the bait position at a specified window length (Fig. 1).

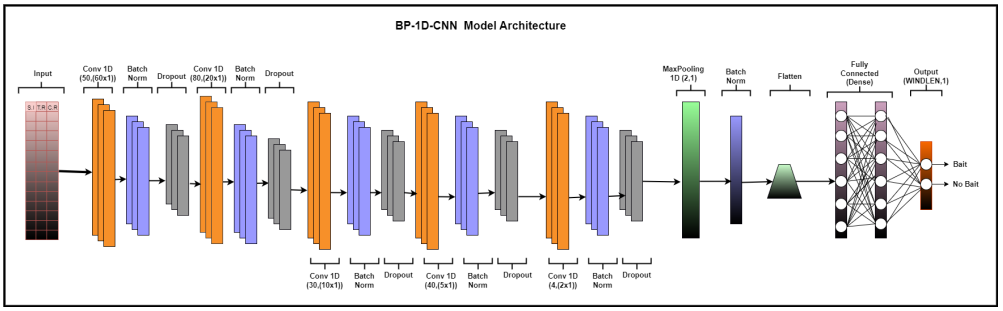

Fig. 1: The proposed CNN model architecture for prediction of bait position.

The input layer receives coverage, on-target, and sequence data information in a defined data segment (window 500 or 1000). The model consists of multiple convolutional layers defined by convolution operation by the equation [2]:

$$y(n) = \sum_{m-\frac{M}{2}}^{\frac{M}{2}} \mathbf{v}(n-m) \cdot \mathbf{u}(n), \tag{1}$$

where $y(n) = v * u$ is the convolution of $\mathbf{v}(\cdot)$ and $\mathbf{u}(\cdot)$, $*$ is the convolution operation, M is the filter size, v and u are input vectors. Each layer is followed by batch normalization and dropout regularization to prevent overfitting.

Here, the first convolutional layer applies 50 filters of size 60 with a striding window of 40. In this layer, we use the ReLU activation function and apply causal padding to preserve the temporal order of the sequence data. Following each convolutional layer, we apply batch normalization to normalize the activation functions of the previous layer to stabilize and speed up the training process.

In addition, we apply additional convolutional layers with different filter sizes and numbers of filters. Each convolutional layer is followed by batch normalization and dropout. Those include: Conv1D (80, (20)) which applies 80 filters of size 20 with causal padding and ReLU activation. Conv1D (30, (10)) which applies 30 filters of size 10 with causal padding and ReLU activation. Conv1D (40, (5)) which applies 40 filters of size 5 with causal padding and ReLU activation. Conv1D (4, (2)) which applies 4 filters of size 2 with causal padding and ReLU activation. After the convolutional layers, we perform the max-pooling defined by the equation:

$$Max\_pool_{v,u}(n) = v_k, \tag{2}$$

where v,u are input vectors, v_i is the i^{th} component of v in which $\mathbf{v_k} \geq \mathbf{v_i}$ for all $\mathbf{n} - \frac{\mathbf{W}}{2} \leq i \leq \mathbf{n} + \frac{\mathbf{W}}{2}$, in which $\mathbf{w} = \frac{Z_i}{\min(Z_i)}$ where $i = 1, 2, \ldots, l$ and $\mathbf{w} \in Z+$ are the filter size of each operation. For our model given $\mathbf{X}$ is the input vector for sequence data (i.e. coverage depth, sequence information, and on target), we defined the max-pooling, $\mathbf{X}_{pool}$ as follows

$$\mathbf{X}_{pool} = \left[Max\_pool_{\mathbf{X},\mathbf{W}}(1), \ldots, Max\_pool_{\mathbf{X},\mathbf{W}}(\mathbf{L}) \right], \tag{3}$$

where $\mathbf{L}$ the output vector size is equivalent to the window size. Here we set max-pooling values with a pool size of 2 and stride of 1. We apply Max pooling to extract the maximum value from each pair of consecutive values in order to reduce the spatial dimensions of the feature maps while preserving important sequence information [3]. The output of the max-pooling layer is flattened into a one-dimensional vector. The output of the max-pooling layer is flattened into a one-dimensional vector. While the proposed CNN architecture includes flattening and pooling layers these can be optionally excluded without significantly impacting the model's predictive performance. Next, a dense layer with several neurons equal to the window size is then added followed by a sigmoid activation function defined by the equation:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{4}$$

where x is the input to the sigmoid function and e is the Euler's number. Since predicting bait position in a given genomic window is a binary classification task. The model computes the probability value for each bait position at a given genomic window. The model then combines these probabilities with the sigmoid values from the output layer in the CNN model. The sigmoid function of CNN model generates values that can be interpreted as the overall probability score of the presence of bait. The score of each predicted bait is determined by the product of its sigmoid activation value and the probability value which is used to assess the biological significance of the predicted bait position in a specified genomic window.

Our model output of the dense layer is reshaped to match the desired output shape of the final prediction of bait position in the input sequence data. To determine the presence or absence of a bait in a given genomic window, we use cross-entropy as a loss function for model training [4]. This is to balance the trade-off between sensitivity and specificity in performance to avoid bias during training. For bait prediction problems with copy number data are a complex task, especially when dealing with overlapping peaks or masked peaks in large genomic segments. For instance, if a certain method tends to call 'no-bait' (i.e. it has a low false-positive error rate), it will show high accuracy, although it misses important baits in a given genomic position. We defined the binary cross-entropy loss function by

$$\ell_i(\mathbf{X}, \theta) = y_i \cdot \log\left(\mathbf{h}_{\mathbf{i},\theta}(\mathbf{X})\right) \mathbf{w} + (1 - y_i) \cdot \log\left(1 - \mathbf{h}_{\mathbf{i},\theta}(\mathbf{X})\right), \tag{5}$$

where $\mathbf{X}$ is the input vector, θ is the set of model parameters, y_i is the i^{th} element of the labeled input data, $\mathbf{w}$ a weight for the importance of false-negative prediction relative to false-positives prediction in the valuation, and $\mathbf{h}_{\mathbf{i},\theta}(\mathbf{X})$ is the i^{th} element in the output predicted for given input $\mathbf{X}$ and model parameter, θ.

For our CNN model to achieve a balance between sensitivity and specificity we set the output vector size (window size), $L = K/2$, and define our final loss function as follows:

$$\mathcal{L}_K(\mathbf{X}, \theta, \mathbf{k}) = \frac{1}{\mathbf{k}} \sum_{n=1}^{k} \ell_{[n]}(\mathbf{X}, \theta), \tag{6}$$

where $\ell_{[n]}(X, \theta)$ is the n^{th} largest individual cross-entropy loss among all l_i and K is number of bait in a given genomic window. The final loss function $\mathcal{L}_K$ was optimized using the Adam optimizer, which uses back-propagation to adjust the model parameters. To enable a direct comparison between actual values and the predictions, we applied causal padding in the 1D CNN layers to preserve a one-to-one correspondence between the input and output layouts. Given that accurate bait prediction likely requires the surrounding genomic and coverage context, we also assessed prediction performance within data segments based on relative position. This approach allows us to account for the impact of flanking regions on prediction accuracy.

References

[1] Mirdita, M., Steinegger, M., Breitwieser, F., Söding, J., Levy Karin, E.: Fast and sensitive taxonomic assignment to metagenomic contigs. Bioinformatics **37**(18), 3029–3031 (2021)

[2] Liu, T., Fang, S., Zhao, Y., Wang, P., Zhang, J.: Implementation of training convolutional neural networks. arXiv preprint arXiv:1506.01195 (2015)

[3] Christlein, V., Spranger, L., Seuret, M., Nicolaou, A., Král, P., Maier, A.: Deep generalized max pooling. In: 2019 International Conference on Document Analysis and Recognition (ICDAR), pp. 1090–1096 (2019). IEEE

[4] Li, L., Doroslovački, M., Loew, M.H.: Approximating the gradient of cross-entropy loss function. IEEE access **8**, 111626–111635 (2020)