Albert-Ludwigs University Freiburg
Department of Computer Science
Bioinformatics Group

Masterproject Report

Improving predictions of Hi-C matrices from ChIP-seq data

Author:
Ralf Krauth

Supervisor:
Joachim Wolff

Submission date:
June 17, 2020

Contents

1 Introduction

The three-dimensional structure of DNA has recently become an active field of research, because it influences many biological processes within living organisms. However, the lab processes required to investigate the so-called chromosome conformation are still comparatively expensive, which has stimulated research in predicting 3D-structure from available data.

One such software package to predict the 3D-structure of DNA, *HiC-Reg*, was conceived by Zhang et al. in 2018/2019 [1, 2]. It is using a random forest regressor to learn DNA conformation from known DNA-DNA- and DNA-protein interactions. Unfortunately, HiC-Reg requires customized input data, produces custom output data and has a high demand for computational ressources, making it inconvenient to use for scientists.

To improve on HiC-Reg's weaknesses, a previous masterproject at the University of Freiburg has resulted in the development of *hicprediction* [3], which is basically a more efficient python implementation of HiC-Reg with standard in- and output formats. However, its predictions are currently not on par with the ones published for HiC-Reg. The goal of this present masterproject is thus to investigate reasons for hicprediction's underperformance and improve the predictions.

1.1 Motivation

Ever since the discovery of the DNA double-helix structure [4] and the central dogma of molecular biology [5] in the 1950's, the vital role of Deoxyribonucleic Acid (DNA) for the existence of all living organisms has been generally known. Less known, however, is the fact that DNA is not only a long molecule holding genetic information, but also forms a spatial structure termed *chromosome conformation*, which facilitates contacts and loops among linearly distant regions in the DNA. Besides the DNA-sequence, the 3D-conformation has a strong influence on important DNA-driven processes like gene expression as well [6, 7].

While techniques to "read" sequences of DNA base pairs are nowadays fast and reliable, methods to determine the spatial structure of DNA are less mature. These so-called *chromosome conformation capture* methods all rely on a chemical fixation of the DNA-structure and some post-processing to count DNA-DNA interactions for certain loci of defined size. The original method from 2002, 3C [8], could only determine interactions between two distinct loci at a time (one vs. one), while more advanced techniques like 4C [9] (one vs. all loci) and 5C [10] (many vs. many loci) have a higher performance. The currently preferred method, Hi-C [11, 12], is capable of determining all DNA-DNA interactions on the genome scale in a single experiment (all vs. all loci). The main output of Hi-C is the so-called contact- or interaction count matrix, which holds the numbers of interactions between all pairs of loci, genome-wide. Hi-C is explained in more detail in subsection 1.2.

Despite the progress in recent years, chromosome conformation capture methods still involve prohibitively expensive and tedious wet-lab processes, which limit their range of applications. In this regard, it must also be noted, that – unlike the DNA sequence –

chromosome conformation is both dynamic and cell-specific, so a large number of experiments is required to get an overview over a whole organism.

Irrespective of the costs, high-resolution Hi-C datasets are already available for several human-, mouse- and drosophila cell lines. Research on these datasets has revealed significant correlations between chromosome conformation and certain proteins bound to DNA, namely specific types of histones and transcription factors [7, 13]. Independent of Hi-C, the corresponding interaction sites between such proteins and DNA can be found with the well-established ChIP-Seq method, a combination of chromatin immunoprecipitation and DNA-sequencing technologies [14, 15], see subsection 1.3. In contrast to Hi-C, this method is fast, affordable, and a plethora of datasets for different proteins and genomes already exists in databases. Researchers are therefore currently investigating possibilities to predict DNA-DNA interaction sites from DNA-protein interaction sites. Conformation capture methods similar to Capture-C [16] could then be employed with a focus on the predicted interaction regions, which would be much more efficient than running Hi-C experiments on the whole genome.

Detecting hidden patterns in large sets of input data and figuring out – potentially non-linear – dependencies between input and output has recently become a domain of machine learning algorithms. For that reason, different machine learning techniques have been applied in the last five years – more or less successfully – to predict DNA-DNA interactions and -structure from available data [2, 17, 18, 19, 20].

During a previous masterproject by Andre Bajorat [3], a regression model based on random forests as proposed by Zhang et al. [2, 1] has been implemented, which permits a direct estimation of contact matrices from ChIP-seq data and genomic distance. The matrices obtained with this approach look promising, but are currently not deemed good enough for general usage.

The goal of this masterproject is therefore to investigate ideas for improving the predictions of the existing machine learning algorithm. This includes discarding, scaling and emphasizing parts of the training sets, concatenating datasets from different cell lines and adding input data with higher peak density than the currently used ChIP-seq narrowPeak files.

However, before going into details, the underlying Hi-C- and ChIP-seq processes shall be explained in more detail, since they are central for understanding the rest of the report.

1.2 Introduction to Hi-C

The Hi-C process is targeted at investigating the three-dimensional structure of DNA by detecting DNA-DNA interactions, as depicted in simplified form in Figure 1.

The typical input to Hi-C are about 20-25 million cells of the same type [12], which are first crosslinked to fix DNA-DNA contacts, e. g. using formaldehyde, and then lysed to extract the DNA. Next, the obtained DNA is cut into fragments by restriction enzymes (1), usually HindIII or DpnII, and the cut ends are filled up with nucleotides partially marked by biotin (2). Blunt ends are then joined (3) under conditions which prefer ligations among open ends over ligations between different fragments, usually achieved by high dilution of the fragments in solvent. The ligated fragments are then purified and sheared into shorter pieces, some of which contain biotinylated nucleotides and some not (4). Those fragments which contain biotinylated nucleotides are then pulled down (5) and subjected to paired-end DNA-sequencing (6). In the end, the outcome of the Hi-C lab process is a bunch of short genomic sequences, so-called reads, which are subsequently processed in the bioinformatics part of the Hi-C protocol.

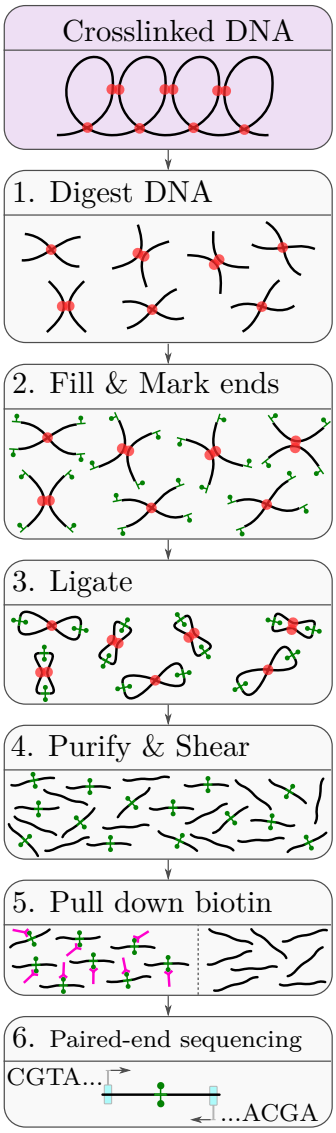


Figure 1: Hi-C lab process

On the bioinformatics side, the reads are first mapped to the relevant reference genome, keeping only so-called chimeric reads, where the sequence from the "left" end of a read uniquely maps to one region of the reference genome and the sequence from the "right" end of the read uniquely maps to another one. These reads are subjected to quality control, and those passing are counted as an interaction between the two regions where both ends have been mapped. The final outcome of a Hi-C experiment is then a (sparse) matrix with interaction counts between all possible pairs of regions. The size of these regions – or resolution of the matrix – depends on the read coverage of the experiment and is often set to 5000 bp, especially in this masterproject.

Often just a small fraction of all reads fulfill the selection criteria outlined above, which makes Hi-C a comparatively expensive process, because several billions of reads may be required to obtain meaningful results [13].

1.3 Introduction to ChIP-seq

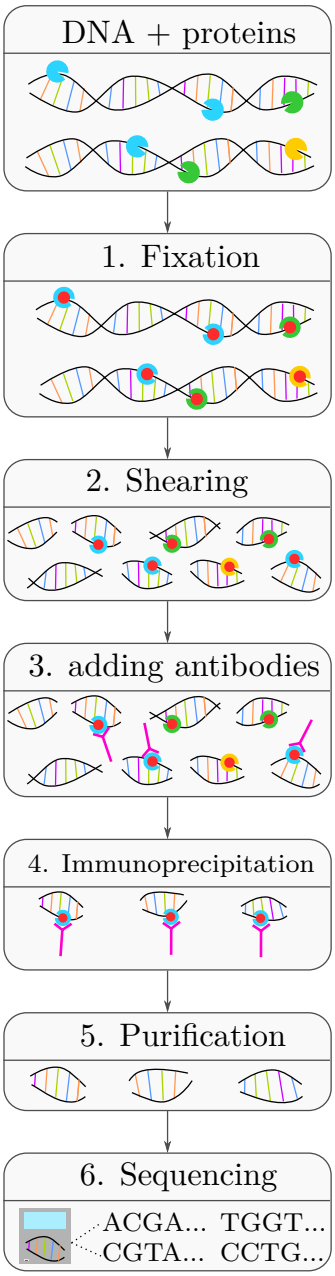


Figure 2: ChIP-seq lab process

The ChIP-seq process is a combination of Chromatin-Immunoprecipitation and DNA-sequencing, designed for investigating DNA-protein interactions.

The typical input to ChIP-seq is a number of cells with DNA that has proteins, histones etc. attached to it, Figure 2, top. As a first step, all these DNA-protein contacts get fixed, e. g. by adding formaldehyde to the cells (1). The cells are then lysed, the DNA-protein structure is extracted and cut into fragments, for example by sonication (2). Next, specific antibodies are added, designed to bind only to a certain protein of interest (3). These antibodies are additionally equipped with a tag, for example a magnetic one, so that the DNA-protein-antibody structures can be precipitated, while fragments without antibodies are discarded (4). The proteins and antibodies are then removed (5), the DNA is purified and finally sequenced (6). Typically, a control experiment is performed together with the ChIP-seq process, which comprises all steps described above, except the immunoprecipitation.

As with Hi-C and all other sequencing-based methods, the outcome of the ChIP-seq lab process is again a bunch of short genomic sequences, which are then fed into the bioinformatics part of the pipeline.

The first data processing step for ChIP-seq, too, is quality control (QC), as with all other sequencing-based processes. Those reads passing QC are next mapped to an appropriate reference genome, to find out where the protein under investigation binds to DNA. Therefore the number of mapped reads per genomic position is counted and compared to the numbers of reads stemming from the control experiment mentioned above. So-called “peaks” are then called at those positions where the number of mapped ChIP-seq reads is statistically significant compared to the control signal. The resulting peaks, i. e. their start-, end- and peak position, along with peak-quality- and strand information, are commonly stored in so-called narrowPeak files. These are simple text files with ten tab-separated columns [21], which currently also serve as input to hicprediction.

2 Prior Work

Having introduced the fundamentals of both processes, it shall now be shown how existing ChIP-seq- and Hi-C data can be used to to predict unknown Hi-C matrices. The following subsections will thus give an overview of the method developed by Andre Bajorat during his masterproject, hicprediction [3], and its foundation, HiC-Reg [2, 1]. This is preceded by a brief introduction into random forests, which are the common basis of both approaches.

2.1 Random forest

Random forests were conceived by Leo Breiman in 2001 [22] and are based on decision trees, probably the most simple approach for supervised machine learning.

Decision trees take as input a set of samples with relevant features and then "learn" to predict a target feature by splitting the input dataset step by step, until the target feature can be decided on. Here, only binary decision criteria on the training features are allowed (True/False), so the decision path can be visualized in form of a tree, see Table 1 and Figure 3.

	train. features		**target feature**
sample nr.	p0	p2	interact. count
0	0.0	0.0	0
...	...	...	...
40000	10.0	10.0	100
40001	10.0	0.0	100
40002	10.0	0.0	100
...	...	...	...

Table 1: excerpt of a training dataset for a decision tree

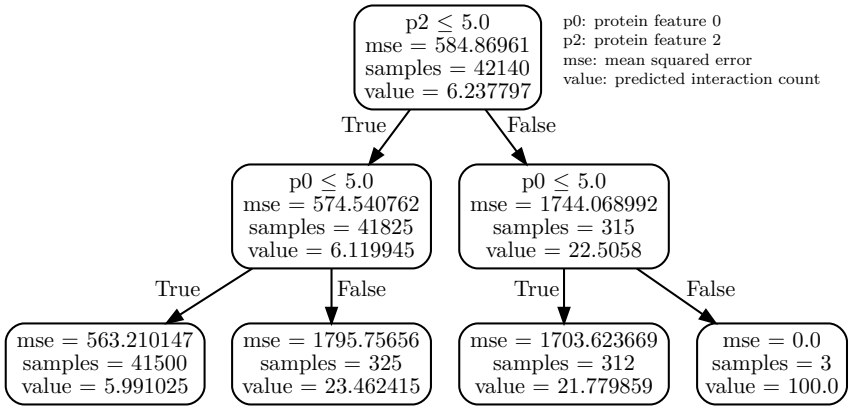


Figure 3: decision tree from hicprediction

Within this masterproject, the target feature is Hi-C interaction count and the training features are the signal values from ChIP-seq data of several proteins, along with the genomic distance of the interacting regions.

Because decision trees are well known for overfitting, i. e. having a low bias but a very high variance, random forests contain not only one, but generally a large number of decision trees. Each tree is trained on a random subset of the original samples, potentially using random subsets of the features for each splitting step. For regression problems, the final prediction is then the average output of all trees, for classification problems usually the majority vote. The rationale here is that random selection of the samples and features reduces the variance, usually at the cost of increased bias. However, in most applications, the reduction in variance more than outweighs the increase in bias.

For regression trees, the cut points, e. g. "p2 $\leq$ 5.0", cf. Figure 3, are usually selected such that the variance in the remaining two datasets – here, all samples with p2 $\leq$ 5.0 on the one hand and all samples with p2 $>$ 5.0 on the other hand – is reduced the most. Searching for the optimal cut-point is computationally involved, but meanwhile there are several open source implementations which handle the task efficiently.

Random forests have a number of tuning parameters. Probably the most important ones are the number of trees in the forest, the size of the subsample used to train each tree, the maximum number of features to investigate for splitting and the minimum number of samples allowed to remain in internal nodes, cf. [23]. All of these are used by HiC-Reg and hicprediction, two random-forest-based methods to predict unknown Hi-C matrices.

2.2 HiC-Reg

HiC-Reg was proposed by Shilu Zhang and colleagues at the Wisconsin Institute for Discovery in 2018 [2, 1]. It takes known Hi-C matrices and ChIP-seq data as inputs to predict unknown Hi-C matrices using random forest regression.

HiC-reg accepts inputs as custom tab-separated text files [24]. For all pairs of (5000 kbp)-regions within a size-adjustable window, the average ChIP-seq read counts of all bins within the window and the distance itself are computed and a so-called WINDOW-dataset is formed from the valid pairs, see Figure 4, upper left and middle. Such a dataset and corresponding Hi-C interaction counts are then merged into a training set for a custom random forest regressor.

After training the regressor, predicting Hi-C interaction counts just requires feeding a test dataset, i. e. processed ChIP-seq data for the target cell line, into the trained random forest, which yields predicted interaction counts as output, Figure 4, middle left. The results, too, are stored in a custom tab-separated text format.

In case data from multiple cell lines is available for training, the window features of all training cell lines can be concatenated into so-called MULTI-CELL test sets, cf. [2, Fig. 1], but these were not used throughout this masterproject. By default, HiC-Reg is using five-fold cross validation on training sets, so it actually trains five different models for each training cell line and chromosome. These models can then be used to predict interactions for a different chromosome in the same cell line (CROSS-CHROMOSOME) or to predict

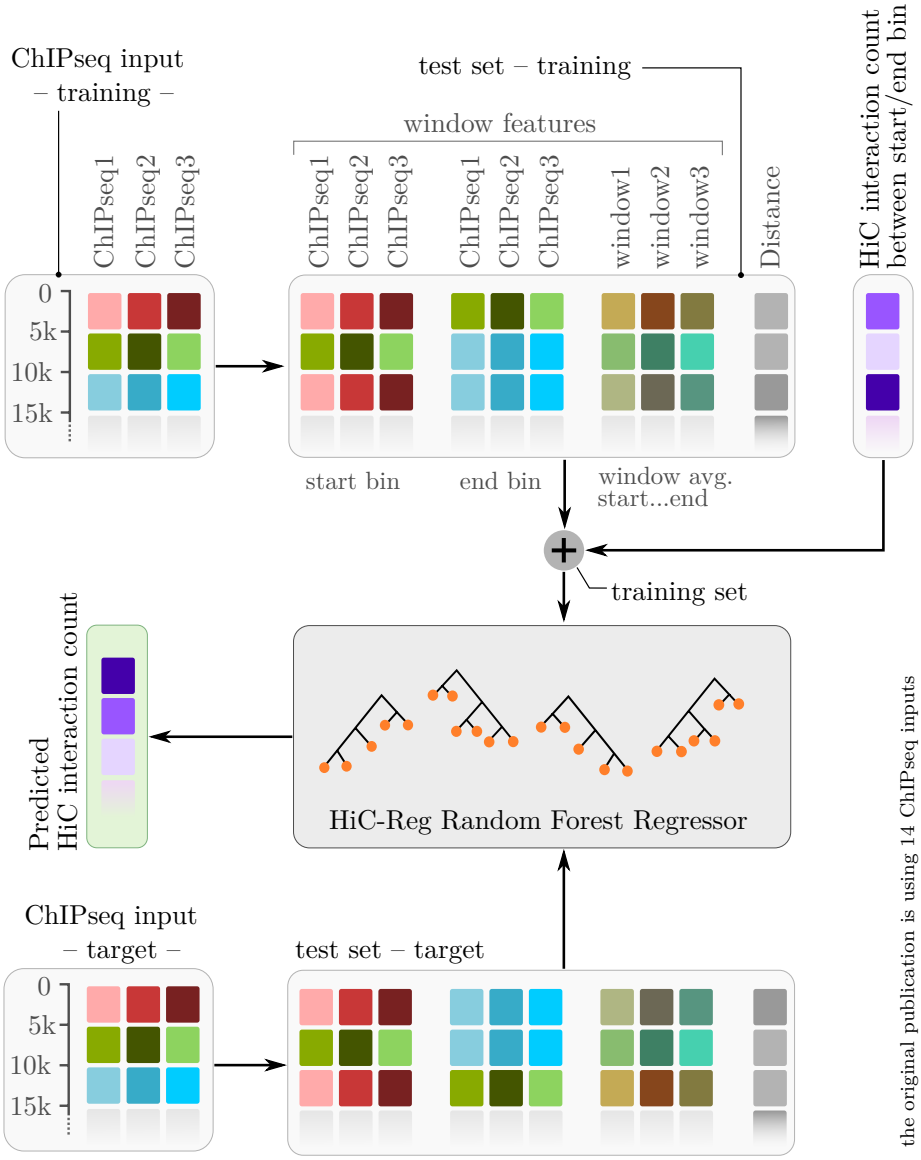


Figure 4: HiC-Reg principle - WINDOW prediction

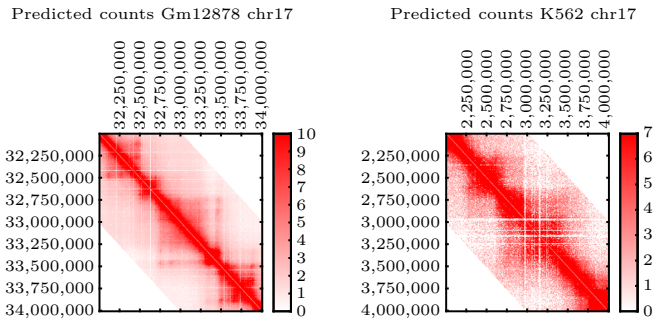


Figure 5: Hi-C matrices predicted by HiC-Reg in MULTI-CELL mode, cf. [2, p. 9]

interactions for the same chromosome in another cell line (CROSS-CELL) – which is more relevant for practical use cases of HiC-Reg. The final predicted matrix is just the average output of the five models.

Despite the aim of predicting Hi-C interaction counts, the publication by Zhang et al. unfortunately contains only two small plots of actual Hi-C matrix snippets, depicted in their original size in Figure 5, and instead relies on statistical measures to assess the quality of the predictions. To this end, the two main metrics used are distance stratified Pearson correlation and the corresponding area under the correlation curve, AUC. While this choice is in line with other relevant publications in the field [17, 18, 19], it seems unfavorable to rely on Pearson correlation too much, since the correlation can be high for Hi-C matrices of biologically unrelated origin [25]. Within this report, correlation plots are thus always used in combination with plots of Hi-C matrix snippets.

Most of HiC-Reg, including the custom random forest regression model, is implemented in C++, complemented by python-, shell- and matlab scripts e. g. for converting and concatenating input data or computing various metrics. The source code and most binaries are available from github [24].

2.3 Hicprediction

Hicprediction was conceived and implemented as part of Andre Bajorat's masterproject at the Bioinformatics Group of the University of Freiburg in 2019 [3]. It is basically a python implementation of the ideas of Zhang et al. outlined above, yet with a few technical differences.

In distinction from HiC-Reg, hicprediction accepts the common file formats narrowPeak and cooler [26] as inputs for ChIP-seq and Hi-C data, respectively, and also outputs Hi-C matrices directly in cooler format. This eliminates the need for text format conversions, but also limits applicability to cases where appropriate input files are available. Another major difference to HiC-Reg is that hicprediction does not perform cross-validation on the training data for CROSS-CELL and CROSS-CHROMOSOME predictions. Instead, it always trains its random forest on the full training set, employing HiC-Reg's WINDOW approach described above. With regard to binning proteins and computing WINDOW-features, hicprediction supports more aggregation methods than HiC-Reg, mean ("avg"), sum and max, see Figure 6. While this might be interesting for future research, sum and max have not been used throughout this masterproject to maintain comparability with the results from Zhang and colleagues.

Hicprediction is made up from four main python modules, which basically implement the workflow depicted in Figure 4. First, *createBasefile.py* [27] aggregates the protein data for training and test into bins of fixed width and stores them in an h5py file, see Figure 6, upper and lower left. Additionally, input Hi-C matrices are cut into separate matrices, one per chromosome, and stored for further usage. Next, *createTrainingSet.py* [28] joins binned protein data with Hi-C interaction counts, computes the WINDOW-features mentioned above and stores the resulting datasets in a compressed binary format (Figure 6, right). The training dataset is then used by *training.py* [29] to train a scikit-learn random forest regressor [30], which is again stored in binary format (Figure 6, middle). Finally,

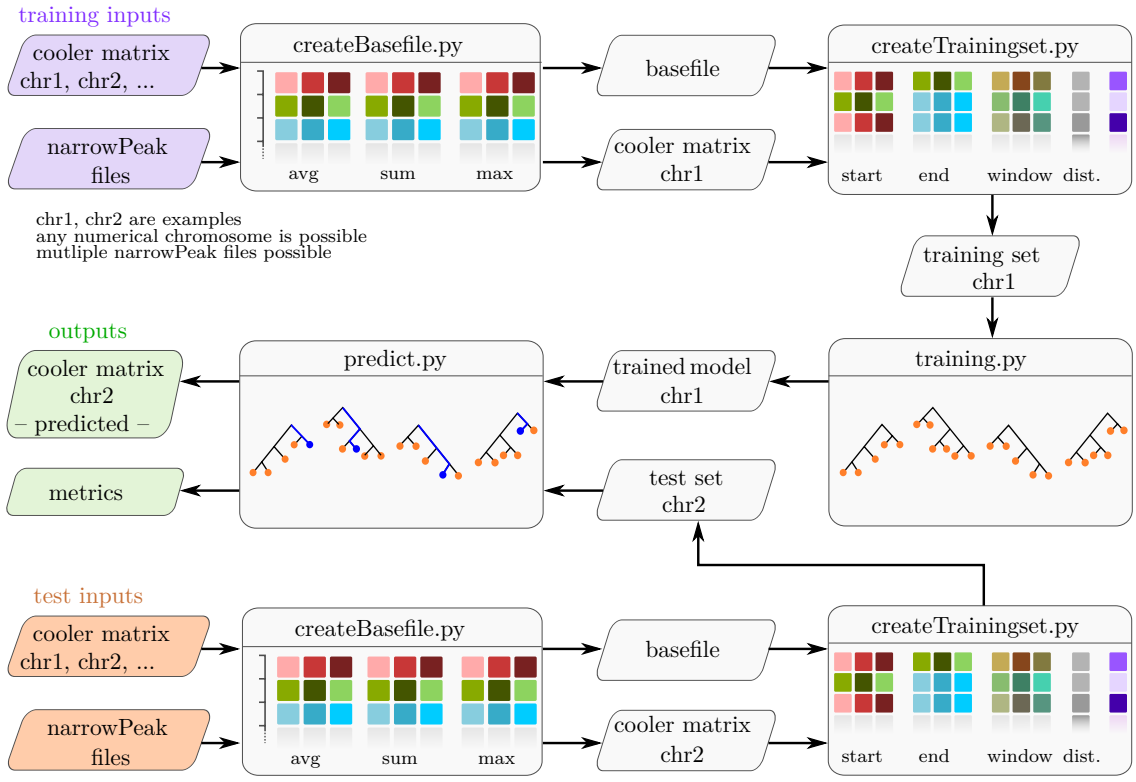


Figure 6: Flow diagram of hicprediction (cross-chromosome prediction)

predict.py [31] takes the test set with protein data from the target cell line, runs it through the trained model and outputs a predicted Hi-C matrix in cooler format (Figure 6, middle left). Additionally, a tab-separated text file can be output, containing distance stratified Pearson correlation values and AUC, plus some other statistical measures and parameter settings for reference.

The original version of hicprediction is available from github [32] and can be installed using conda [33]. Each of the four python modules mentioned above takes a number of options and further inputs, which are described in detail on the readme page in github.

3 Ideas to improve hicprediction

Despite noticeable progress in recent years, no machine learning method is currently able to predict flawless Hi-C matrices, hicprediction being no exception. The following subsections will thus investigate some of its shortcomings and present ideas to improve on them.

While the reasons for erroneous predictions can be manifold, this report mainly focuses on filtering, rectifying and enhancing input data. Additionally, a different machine learning model for hicprediction – extra trees – will be investigated.

3.1 Avoiding training samples without protein data

In the initial results from hicprediction, at least two issues are obvious, Figure 7: The gradient-like regions (A) and the ubiquitous inverted "triangles" (B) in the prediction.

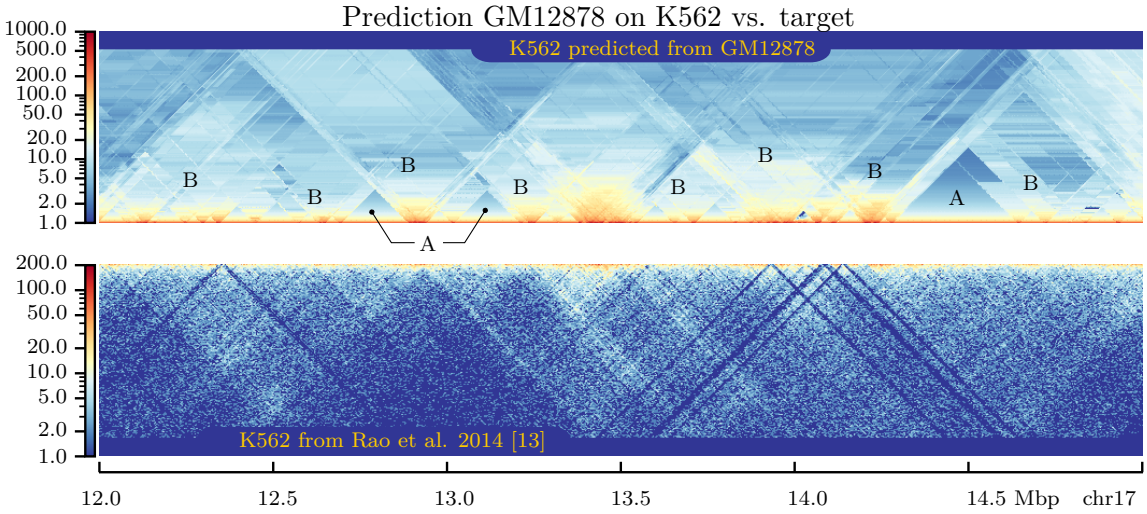


Figure 7: Example prediction from hicprediction with gradients and inverted triangles

The causes of these errors shall be investigated by means of a simple example. Assume a training situation as in Figure 8a, which comprises one region with a lot of interactions, a large remaining area without any interactions and three ChIP-seq peaks from a single protein distributed within and delimiting the interacting region. From that, it should be easy to predict the target situation in Figure 8b, since the latter is just the training situation shifted 1 Mbp to the right. However, even in such a simple case, the predicted matrix from hicprediction is erroneous and contains both gradient-like regions and a number of inverted triangles, Figure 8c. What looks confused at the first glance is actually to a large extent well explainable, if one considers the values of the features for different regions of the training set and recalls what the algorithm could learn from them, Figure 9a and Table 2.

Gradient-like predictions are most prevalent in those regions where distance is the only feature available for predicting, Figure 9b ($\alpha + \beta$), corresponding to training regions (A) and (B), Figure 9a. To understand this fact, it helps to recall how decision trees can split

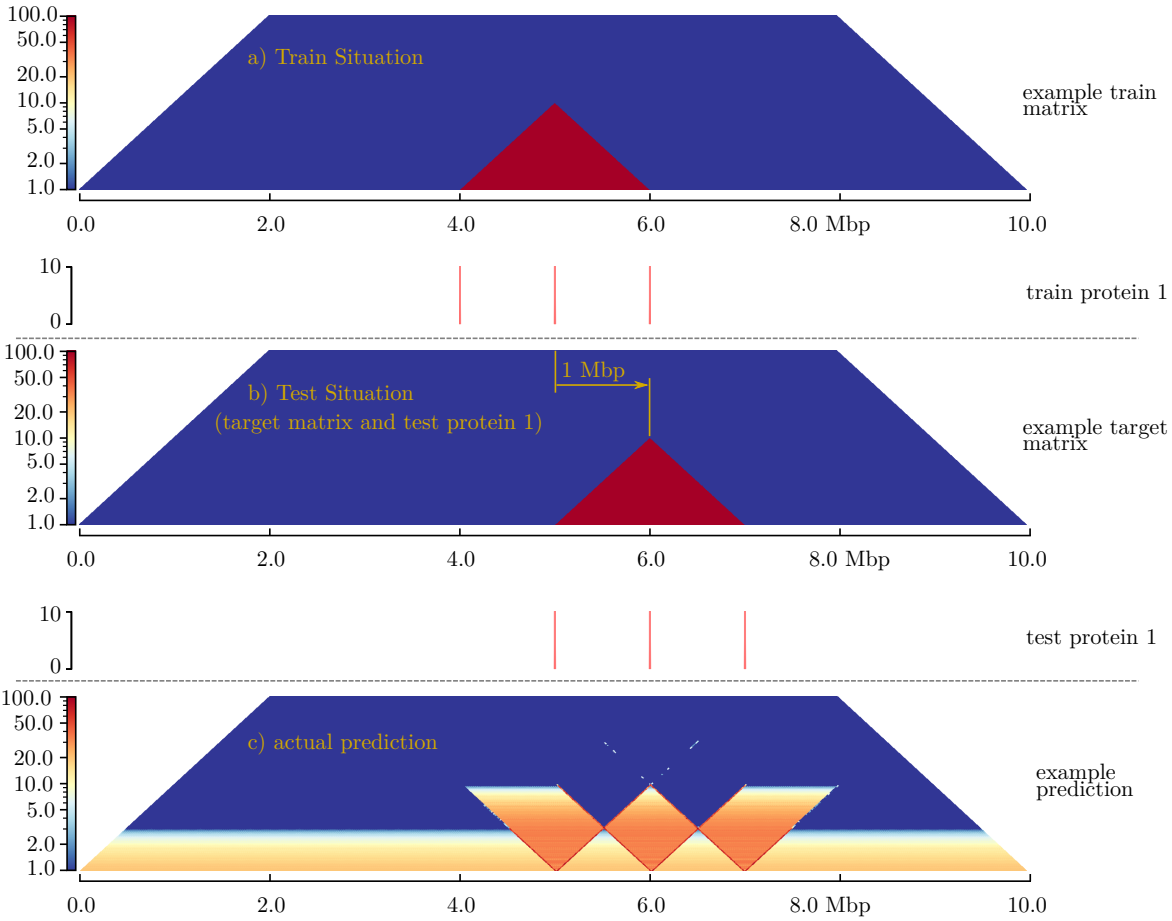
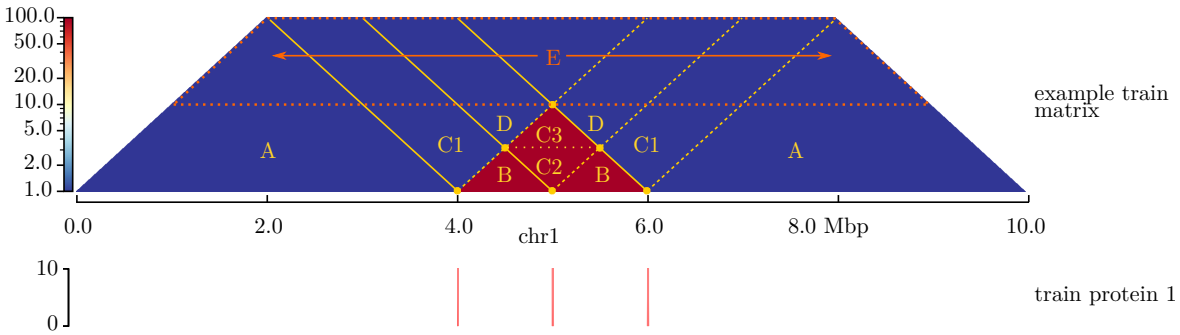


Figure 8: example with inverted triangles and gradient-style predictions

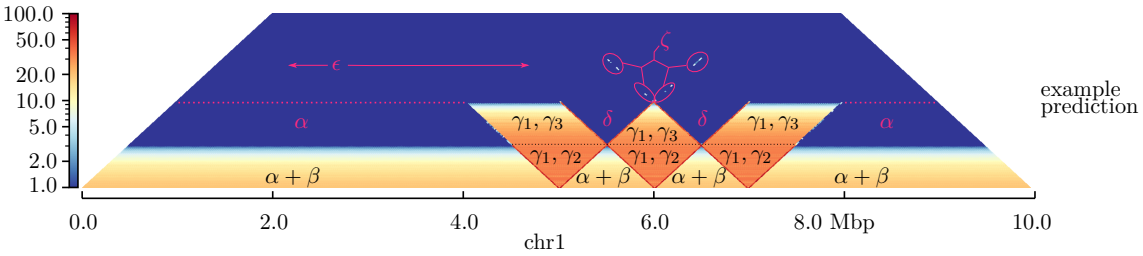
the datasets in such distance-only cases. For a distance of 1 bp for example, there are approximately four times more training samples in region (A) with a target read count of zero than in region (B) with a read count of 100. The resulting predictions for such samples are then around the mean, $\frac{4 \cdot 0 + 1 \cdot 100}{5} = 20$. For a distance of 500 000 bp, there are about 8 times more samples in region (A) than in (B), causing predictions of around $\frac{8 \cdot 0 + 1 \cdot 100}{9} \approx 11.1$. For distances greater than 1 Mbp (and all other features still zero), there are no more samples in region (B), so the prediction relies on training samples from region (A), which are all zero, Figure 9b (α) and (ϵ).

For distances greater than 1 Mbp (and all other features still zero), the prediction relies on training samples which all have zero interaction count, Figure 9b (α) and (ϵ).

The inverted-triangle predictions from Figure 8c occur in situations where distance and window protein are the only available training features, Figure 9b, (γ_1), (γ_2), (γ_3) and (δ), corresponding to training regions Figure 9a (C1), (C2), (C3) and (D). Again, it helps to look at different distances. Between 1 bp and 1 Mbp, there are constantly about twice as many training samples in region (C1) than in (C2), with target read counts of 0 and 100, respectively. The resulting prediction is thus about $\frac{2 \cdot 0 + 1 \cdot 100}{3} \approx 33.3$, Figure 9b, ($\gamma_1$, γ_2). For distances greater than 1 Mbp, the number of samples in region (C1) grows linearly,



(a) learning regions for the random forest



(b) prediction regions

Figure 9: explanation approach for inverted triangles and gradient-style predictions

while the number of samples in region (C3) shrinks linearly, causing a gradient prediction within the inverted triangles, Figure 9b (γ_1, γ_3). Region (D) is distinguishable from regions (C1) to (C3) by the window feature value, which is always twice as large in region (D) for the same distance, because the window contains two protein peaks. Since all training samples within region (D) are zero, the corresponding regions (δ) in the prediction are also zero, causing the inverted-triangle-shaped "cut-outs" in the prediction, Figure 9b.

In reality, the learning process is of course more complicated than described above, because not all features are taken into account at each split, cf. subsection 2.1, so that some trees put more weight on features like start and end. This is probably the reason for the few outliers in the prediction, Figure 9b (ζ), which cannot directly be explained from the learning regions above. Across all trees, distance is still by far the most important feature, Figure 10, followed by the window feature and lastly by the equally (un-)important start and end features.

Probably the most simple way to prevent the random forest from learning gradients and inverted triangles is discarding all "empty" training- and test samples, i. e. samples which have zero signal value for all proteins, cf. subsection 4.4. The results of this approach are shown in subsection 5.1. However, discarding empty samples usually leads to sparse training- and test sets, causing other types of problems. These will be handled below in subsection 3.2.

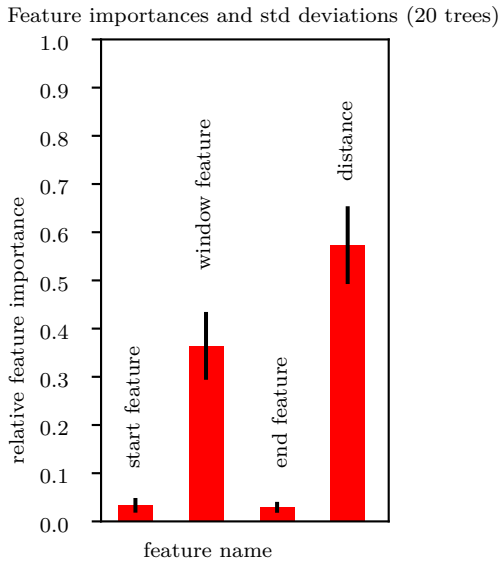


Figure 10: example feature importances

region	**potential decision rules learned**
A	When all features except *distance* are zero, the *target value* is 0
B	When all features except *distance* are zero and $distance \leq 1\,\text{Mbp}$, the *target value* is 100
C1	When $window\ feature = \frac{10}{distance}$, the *target value* is 0
C2	When $window\ feature = \frac{10}{distance}$ and $distance \leq 1\,\text{Mbp}$, the *target value* is 100
C3	When $window\ feature = \frac{10}{distance}$ and $distance > 1\,\text{Mbp}$, the *target value* is 100
D	When $window\ feature = \frac{2 \cdot 10}{distance}$, the *target value* is 0
E	When $distance > 2\,\text{Mbp}$ or $window\ feature = \frac{3 \cdot 10}{distance}$, the *target value* is 0
yellow solid lines	When *end feature* nonzero, the *target value* is between 0 (about 80% of samples) and 100 (about 20% of samples)
yellow dashed lines	When *start feature* nonzero, the *target value* is between 0 (about 80% of samples) and 100 (about 20% of samples)
yellow circles	When *start-* and *end feature* nonzero, the *target value* is 100

Table 2: learning regions for the random forest

3.2 Dealing with sparse input data

As already mentioned in subsection 1.3 and Figure 6, hicprediction thus far relies on narrowPeak-files for incorporating ChIP-seq data, cf. subsection 4.2. Compared to the length of the single chromosomes, the number of peaks contained in these files is often small. For example, in K562, there are 2180 ChIP-seq peaks in the CTCF-narrowPeak file for chromosome 17, while the length of the chromosome is 81 195 210 bp[1] – roughly one peak every 37 246 bp. After binning at 5 kbp resolution, only 1943, or 11.96% of 16 240 CTCF bins are nonzero in K562, cf. Table 3.

protein	GM12878		HMEC		HUVEC		K562		NHEK	
	pk	bw	pk	bw	pk	bw	pk	bw	pk	bw
CTCF	10.39	94.86	9.01	94.77	9.75	94.36	11.96	94.42	10.68	93.85
DNaseI	29.53	93.95	34.05	93.80	28.25	93.61	27.03	94.28	31.38	93.94
H3K4me1	9.06	93.83	9.48	94.52	9.01	94.05	8.34	94.33	10.39	93.23
H3K4me2	2.51	94.54	2.08	94.78	6.45	94.30	6.70	94.44	7.07	94.20
H3K4me3	11.06	94.55	8.23	94.66	7.92	94.18	9.76	93.93	12.60	93.44
H3K9ac	12.19	94.59	17.34	94.90	14.53	94.04	15.14	94.16	18.28	93.53
H3K9me3	12.64	94.20	13.78	94.73	10.92	94.08	11.07	94.14	15.25	93.89
H3K27ac	7.78	94.77	8.90	94.61	6.91	93.96	7.39	94.42	8.86	94.22
H3K27me3	8.95	94.83	6.96	94.95	6.50	94.85	7.19	94.83	6.88	95.26
H3K36me3	7.16	94.33	7.37	94.74	7.77	94.17	6.64	94.48	9.61	94.34
H3K79me2	3.75	94.95	8.19	94.76	7.57	95.21	1.03	95.14	10.95	95.33
H4K20me1	2.42	94.56	1.75	95.06	1.77	94.60	5.21	94.53	3.50	94.22

total 16240 bins, pk: narrow-/broadPeak, bw: bigwig

Table 3: percentage of nonzero bins for chr17 at 5 kbp resolution

Even with the 12 proteins used throughout this masterproject, the training sets remain quite sparse. When discarding empty samples to avoid inverted triangles and gradient-style predictions, often only few valid samples remain in the datasets, causing fragmented predictions, Figure 15.

To reduce the number of empty ranges in the predictions, two simple techniques were tried first, *filling empty areas in the predicted matrix* using a two-dimensional Gaussian kernel and *filling empty bins in the inputs* using a one-dimensional Gauss filter on each of the narrowPeak Files. While the first filtering approach is mitigating the symptoms, the second is addressing the underlying problem of sparse input data directly. Both smoothing techniques could in principle also be combined, but this has not been investigated.

Another option for dealing with sparse inputs is coarsening the resolution of all Hi-C matrices to 25 kbp. This leads to a five times smaller number of bins in every chromosome, reducing the probability of empty bins and thus the sparsity of the input data.

Since the three approaches mentioned above either did not yield satisfying results or involved a loss of resolution, another approach directed at generating more dense inputs was made. The reason for the narrowPeak-files being sparse is that they contain peaks

[1] reference genome hg19

only at those genomic positions were a certain protein is most likely to interact with the appropriate DNA. However, it is also possible to directly use the (normalized) number of mapped ChIP-seq reads per genomic position for the protein of interest, the so-called read coverage, and let the random forest learn the relation between read coverage and interaction count itself. For convenience, the read coverage was computed from the mapped reads in BAM-format and stored in so-called bigwig files, see subsection 4.2. It is obvious from Figure 11 and Table 3 that these files cover more genomic positions than the corresponding narrowPeak files, but they may also have the disadvantage of including background noise.

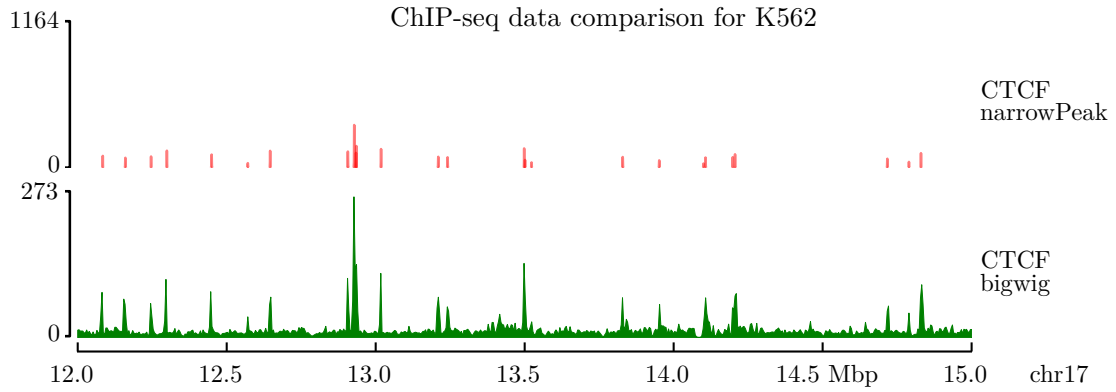


Figure 11: ChIP-seq data for CTCF from bigwig and narrowPeak files

This is because they usually contain reads from genomic positions where the proteins of interest were found only accidentally, e. g. because they were floating around these positions when the fixation step of the ChIP-seq process (Figure 1, top) was performed.

The results for the two filter-based approaches and predictions from bigwig files at 5 and 25 kbp resolution are shown in subsection 5.2.

3.3 Scaling protein signal values and Hi-C interaction counts

Remarkably, no matter which types of input files were used thus far, the absolute values of the predicted Hi-C interaction counts were usually not in the target range, and sometimes off by factors of 5 or more, see e. g. Figure 19. Such discrepancies can arise when the magnitude and distribution of the values in training samples differ from the target – which is often the case for the cell lines and chromosomes used throughout this masterproject. For example, when predicting K562 from GM12878, the random forest is trained on interaction counts from GM12878, which are mainly in the range [0...4000], Figure 12 top panel. As the algorithm has seen many training samples within this value range, it overestimates the interaction counts from K562, which are actually mostly in the smaller range [0...800], Figure 12 bottom panel. To circumvent this, all Hi-C matrices were scaled to the same fixed value range on a per-chromosome basis. Four ranges were tried, [0...1], [0...10], [0...100] and [0...1000]; the results are shown in subsection 5.3.

Looking at the protein data, it can be seen that the absolute read coverage values (signal values) also vary significantly among the 12 inputs of each cell line, see e. g. Figure 13.

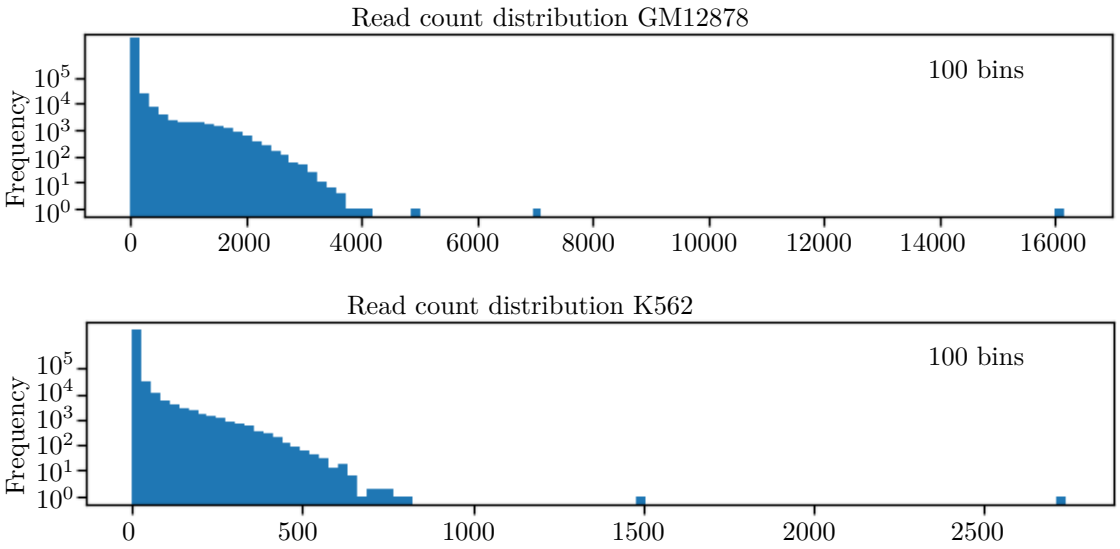


Figure 12: Hi-C interaction count distributions for chr17

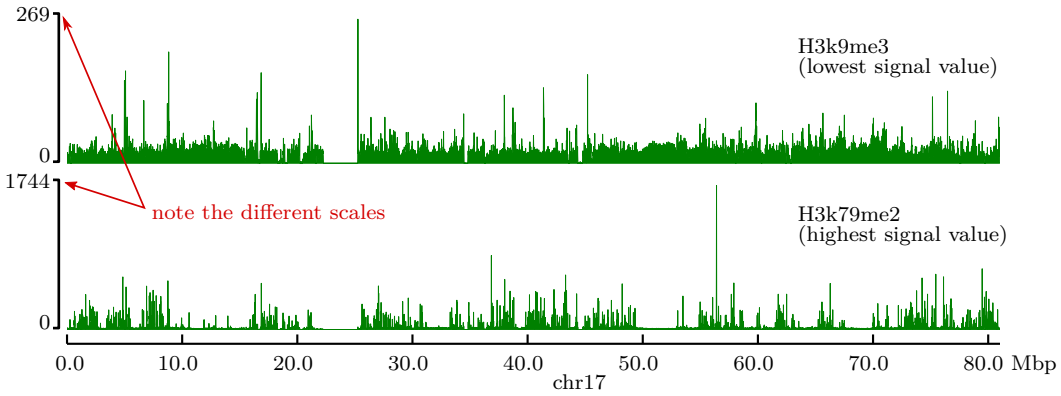


Figure 13: bigwig signal value comparison GM12878, chromosome 17

The same holds for the signal values of narrow- and broadPeak files. The signal values were thus again scaled into the same fixed ranges as above, on a per-chromosome basis.

In a further test, all protein signal values were scaled by dividing them by their respective mean. This somewhat unorthodox way of scaling was found commented out in the code by Zhang et al. [24]. Division-by-mean scaling reduces the value ranges of all proteins, as the mean values of the protein signals are all greater than one, but does not guarantee a uniform value range, Figure 52a and 52b. For chromosome 17, it turned out that the divisors differed from protein to protein, but within a single protein, they were often roughly equivalent across all five cell lines, Figure 53. Note that a unified range could be achieved by subsequent value range scaling, but this was not tested.

Feature- and interaction count scaling should theoretically not affect the performance of hicprediction. Pearson correlation is invariant to scaling by definition, and random forests, unlike many other supervised learning techniques, are also impervious to different value

ranges of the features. The latter holds because split points for the decision trees are only computed for a single feature at a time, compare subsection 2.1. For example, if the optimal cut-point was 0.8 for a certain feature with value range [0...1], it would simply be 8.0 for the same feature scaled to range [0...10], and the cut-point would be no different, if another feature had value range [0...5] or [20...1000]. However, adjusting value ranges should be useful for concatenating datasets from different cell lines, see below, and can help avoid problems interpreting feature importances [34].

3.4 Concatenating datasets and predictions from different cell lines

In the previous sections, only training data from single cell lines have been used at a time to predict interactions in another cell line. However, as the cell lines used within this masterproject have different biological functions, they are likely to have alternative 3D-conformations, too. This means that different relations between proteins and Hi-C interaction counts can be learned from each cell line, so that the choice of the training cell line can make a significant difference.

To reduce the dependency on the choice of the training cell line on the one hand and make full use of all available training data on the other hand, two approaches were investigated. Firstly, it is obviously possible to train models separately for all (four) available cell lines, sum up the predicted matrices and compute the mean. This is somewhat similar to the ENSEMBLE approach by Zhang et al. [2] and can easily be accomplished using existing scripts from `deeptools`, see subsection 4.6. Secondly, although the MULTI-CELL approach from [2] is not yet implemented, it is still possible to create a joint dataset by simply concatenating the datasets from the single training cell lines into one larger dataset. This joint dataset can then be used as input to the existing training- and prediction modules of hicprediction.

The results of both concatenation approaches are to be found in subsection 5.4.

3.5 Emphasizing certain samples

Revisiting the read count distribution in Figure 12, it is obvious that the vast majority of training samples will have an interaction count close to zero. The random forest is thus likely to associate the protein inputs of the corresponding training samples with small interaction counts. However, structures like TADs should correspond to samples with nonzero, hopefully comparatively high interaction counts, which seem underrepresented in the training data. In total, three options to improve on this were investigated. First, samples with "high" Hi-C interaction counts were given higher weights to make them more relevant when the splits for the single decision trees were computed. Second, samples within certain TADs were given higher weights, and finally, samples with "high" CTCF signal value were given higher weights. The last method is based on the idea that CTCF is known to be enriched in TADs [35], and determining samples with "high" CTCF signal value is easier than finding TADs.

A problem common to all methods mentioned above is that the choice of the samples to emphasize is not straightforward – for example, there is no obvious way to define "high"

CTCF signal values. Within this report, three parameters were used to select and weight samples: *lower* and *upper* for selecting samples based on their CTCF signal value and interaction count, respectively, and a factor k, which allows to control the weights of the selected samples in relation to the weights of the non-selected ones. To find optimal values for the three parameters, a search strategy based on tree-structured Parzen estimators was followed, cf. subsection 4.7. For TAD-based weighting, TADs were called with an established algorithm and samples within TADs were weighted using a factor k as above, see subsection 4.7 for details.

One further approach, which is in a similar spirit as the weight-based ones above, was again borrowed from Zhang et al. [2]. Although apparently not documented anywhere in the paper, the supplementary data as well as the two small plots of predicted matrices, Figure 5, suggest that interacting pairs with a distance of less than 5000 bp have been discarded from the HiC-Reg datasets. Since the highest interaction counts typically occur at the lowest distances, this way of pruning the datasets predominantly discards samples with high interaction counts and thus implicitly gives more weight to samples with *lower* interaction counts. As a side effect, outliers with very high interaction counts are often also removed, because they commonly stem from the main- or first side diagonal of the matrix, i. e. have a distance less or equal to 5 kbp.

The results of all sample-emphasizing methods are to be found in subsection 5.5.

3.6 Replacing random forest by extra tree regression

Extremely randomized trees, or extra trees, are an advancement from random forests [36]. They have the same adjustment parameters as random forests and are thus a natural choice to be investigated.

The main difference between the two models is that the computation of the cut-points is randomized in extra trees, while random forests always search for the optimal cut-point in the given (sub-)set of features. Additionally, extra trees typically draw from the training datasets without replacement, while random forests use bootstrapping by default. Both changes are meant to reduce variance while maintaining a low bias [36].

The results for all improvement ideas discussed above, as well as a concluding comparison between (improved) hicprediction- and HiC-Reg results will be shown in section 5, while the following section 4 will provide implementation details for the methods introduced above.

4 Methods

In the following subsections, implementation details are provided for the measures to improve hicprediction described in section 3, preceded by a description of input data preparation. Finally, the approach for comparing hicprediction and HiC-Reg is discussed.

4.1 Hi-C matrices and matrix plots

As in Zhang et al. [2], Hi-C matrices of five well-investigated training and test cell lines were taken from Rao et al. [13]. All matrices were downloaded directly from the gene expression omnibus, accession GSE63525, in .hic-format, see Table 4. In all cases, the "combined_30" datasets were selected, which contain only high-quality read pairs from primary and replicate.

cell line	file name	link
GM12878	GSE63525_GM12878_insitu_primary+replicate_combined_30.hic	[37]
HMEC	GSE63525_HMEC_combined_30.hic	[38]
HUVEC	GSE63525_HUVEC_combined_30.hic	[39]
K562	GSE63525_K562_combined_30.hic	[40]
NHEK	GSE63525_NHEK_combined_30.hic	[41]

Table 4: File names and links for Hi-C matrices

All Hi-C matrices were then converted to cooler format using `hic2cool convert`, version 0.7.1. This was done twice, for 5000 bp and 25 000 bp resolution, using `hic2cool`'s `-r` option with values of 5000 and 25 000, respectively. For HMEC, HUVEC and NHEK, 5 kbp is the highest available resolution.

All heatmap plots of Hi-C matrices within this report were made with `pyGenomeTracks` setting a width of 15 cm at 200 dpi resolution (`-width 15 -dpi 200`), using the log scale plus one, "log1p". As is common for Hi-C matrices, only the upper or lower triangular part is shown, rotated counterclockwise by 45°, so that the diagonals become horizontal lines. The plotted region is given at the bottom of each plot – here, chromosome 17, 12...15 Mbp was often used, because this region originally featured many "inverted triangles" and "gradient-style" predictions, see Figure 7. Any improvements made to the predictions were thus thought to appear particularly obvious in this region. The color coding of the plots was approximately scaled to the actual value range in the plotted region, and not to the value range of the full matrix. This is advantageous for identifying structures in the plots, but also means that different plots as well as different panels in the same plot may have different scales and color coding. To ensure comparability with the data from Zhang et al. [2], only interacting regions with a distance below 1 Mbp were considered for hicprediction, and the plots were limited to 1.1 Mbp.

4.2 ChIP-seq data

In their publication on HiC-Reg [2], Zhang and colleagues used data from 14 proteins and histones, which they partially downloaded from ENCODE and partially feature-engineered, see table Table 5 for a list of all features used.

		hicprediction		
protein	HiC-Reg	npk	bpk	bigwig
CTCF	+	+	-	+
DNaseI	+	+	-	+
H3K27ac	+	+	-	+
H3K27me3	+	+	-	+
H3K36me3	+	+	-	+
H3K4me1	+	+	-	+
H3K4me2	+	+	-	+
H3K4me3	+	+	-	+
H3K79me2	+	1)	2)	+
H3K9ac	+	+	-	+
H3K9me3	+	1)	2)	+
H4K20me1	+	+	-	+
RAD21	+	-	-	-
TBP	+	-	-	-

npk: narrowPeak
bpk: broadpeak
1) GM12878, K562
2) HMEC, HUVEC, NHEK

Table 5: proteins and histones for HiC-reg and hicprediction

For hicprediction, the relevant narrowPeak files of the proteins and histones in Table 5 were downloaded from ENCODE/Broad Institute `http://ftp.ebi.ac.uk/pub/databases/ensembl/encode/integration_data_jan2011/byDataType/peaks/jan2011/spp/optimal/` and unpacked into separate folders, one per cell line. DNAse data was downloaded from `https://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeOpenChromDnase/`. Next, the 12 files in each of the five folders were renamed such that the lexicographic order of the files, and thus the order of processing in hicprediction, was the same within each folder.

To enable working with higher-density input data, BAM files containing mapped reads from ChIP-seq experiments were downloaded from `https://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeBroadHistone/` (proteins, histones) and `https://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeOpenChromDnase/` (DNaseI). The files were then indexed using `samtools` version 1.9 and aggregated into bigwig files using `bamCoverage` version 3.3.1. These steps were done for replicate 1 and 2 of each ChIP-seq experiment. Next, the mean read coverage was computed from each pair of replicates using `bigwigCompare` version 3.3.1, resulting in a single bigwig file per cell line per protein/histone/DNaseI. The full commands and their corresponding options are given below in Listing 1 (p. 68). Finally, the bigwig files were renamed as described above.

Bigwig files can efficiently be queried for aggregated read counts from selected regions and are comparatively small when stored at a resolution of 5 kbp, which allows fast and simple protein binning. For all ChIP-seq experiments in this masterproject, bigwig files could also have been downloaded directly from the sources mentioned above. However, it proved difficult to find out how exactly these files were created, so this was not done.

In HMEC, HUVEC and NHEK, ChIP-seq data for H3k79me2 and H3k9me3 are not available in narrowPeak, but instead in broadPeak format. This is format-wise essentially the same as narrowPeak without the "Peak" column, i. e. without the position of the called peaks. BroadPeak files were downloaded from `https://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeBroadHistone/`, unpacked and renamed as described above. Hicprediction was then modified to internally convert the broadPeak files to narrowPeak, taking $\lfloor \frac{start+end}{2} \rfloor$ as peak position. It is possible to use narrowPeak-, broadPeak- and bigWig-files simultaneously in hiprediction, provided the same order of the inputs is maintained across all cell lines and chromosomes, cf. subsection 5.2.

For RAD21 and TBP, no BAM-files and neither of the three usable file formats bigwig, narrow- or broadPeak could be found for cell lines HMEC, HUVEC and NHEK. This is probably because no ChIP-seq experiments have been published for those two proteins in these three cell lines so far. RAD21 and TBP were therefore exluded from hicprediction to maintain compatibility among the datasets from different cell lines.

Text files containing the download links for all protein- and histone files in BAM- and narrow-/broadPeak format can be found in the github repository [42]; `wget -i` can be used with these to conveniently download all relevant files for each cell line at once.

ChIP-seq data in bigwig- and narrowPeak format were plotted with pyGenomeTracks, the bin number of the bigwig files being adjusted according to the bin size of the corresponding Hi-C matrices.

4.3 Random forest and Extra trees implementation

Hicprediction's random forest implementation is imported from sklearn.ensemble [30, 23]. All parameters are adjustable from the command line, but only the default values in Listing 2 (p. 68) were used for the investigations in this report, compare *training.py* in the github repository [42].

The number of trees, `n_estimators`, and the `max_features` parameter, which controls the number of features drawn at random for each split, were chosen to match the corresponding settings of HiC-Reg. All other parameters are either sklearn defaults or were set to values deemed reasonable for the task at hand.

The extra trees algorithm was also imported from sklearn.ensemble, using the same parameters as shown in Listing 2 for the random forest, except for `bootstrap`, which was set to `False`. To allow handling random forests and extra trees in the same module, the boolean parameter `-useExtraTrees` has been added to *training.py* to select the desired algorithm, see the hicprediction github for details [42].

4.4 Investigations on discarding empty samples and input density

The example training- and test matrices for investigations on empty samples in subsection 3.1 were created using a small python script, *createFakeMatrices.py*, which is available from the masterproject github repository [42]. The bin width (matrix resolution) was set to 20 000 bp and the maximum distance to 4 000 000 bp, which yields 79 900 training samples. The "interacting region" was adjusted to be 2 000 000 bp wide, which corresponds to 5050 training samples, each with a read count of 100. The ChIP-seq data was entered into hicprediction as narrowPeak files, these were simply handmade with a text editor. Each peak was 200 bp wide with peak positions at 4 000 000 / 5 000 000 / 5 999 999 bp for training and 5 000 000 / 6 000 000 / 6 999 999 bp for test. The signal value was 10 in all cases.

For protein input smoothing, standard functions from pandas [43] were used to apply gaussian kernels, in this case truncated at four times the standard deviation (but at least one bin on either side), see *createTrainingSet.py* for details [42]. The standard deviation σ was manually adjusted between 0.1 and 10 until acceptable results were obtained; the values used in subsection 5.2 are stated in the respective figures.

For matrix smoothing, the `gaussian_filter` function from scipy.ndimage [44, 45] was used, again manually adjusting the standard deviation, see *predict.py* [42]. This filter was also truncated at four times the standard deviation.

Two different options for discarding "empty samples" without start- and end-feature were considered, termed *single-protein-rectification* and *any-protein rectification*. Single-protein rectification means keeping only samples in the datasets where at least one protein has nonzero start- and end-features. Any-protein-rectification on the other hand keeps samples where at least one start- and at least one end-feature are nonzero, no matter from which of the proteins. It is obvious that any-protein-rectified datasets are in general a superset of single-protein-rectified datasets, because they contain all samples from single-protein rectification and additionally such samples where at least two different proteins have nonzero start- and end feature, but no single protein has both nonzero start- and end-feature. These additional samples all have zero window features, while the single-protein approach keeps only – but generally not all – samples with at least one nonzero window feature.

Since it seemed impossible to conclude theoretically whether the single- or the any-protein-method is better, the decision was made empirically. To this end, CROSS-CELL predictions were conducted from GM12878 on K562 at 25 kbp resolution on chromosomes 9 and 17 using both approaches. Across all tested datasets, the difference in the sample numbers was typically small. The predictions were thus highly similar, with slightly better results for any-protein rectification in some cases. All investigations in this report that involved removing empty samples were therefore made using any-protein rectification.

Rectification was always done on both training- and test set to prevent the algorithm from learning invalid decision criteria in areas with insufficient input data. Positions in the target matrix which could not be predicted due to unavailability of inputs were left unset in the sparse matrix data structure, corresponding to zero interaction count. Distance stratified Pearson correlation was always computed including these "trivial" zero counts.

4.5 Scaling interaction counts and protein signal values

The interaction counts were scaled to value ranges [0...1], [0...10], [0...100] and [0...1000] according to Equation 1, where $sampleMin$ and $sampleMax$ are the minimum and maximum value of a certain feature across all samples in a given per-chromosome dataset. It should be noted that for typical Hi-C matrices, $sampleMin$ is zero and $targetMin$ was set to zero, so that, with regard to interaction counts, eq. 1 is effectively a multiplication.

$$scaledValue = targetMin + \frac{actualValue - sampleMin}{sampleMax - sampleMin} \cdot (targetMax - targetMin) \quad (1)$$

For convenience, value range scaling was performed using the MinMaxScaler class from the sklearn.preprocessing module. To suppress noise, values below 0.001 were set to zero after scaling, except for computations with value range [0...1], where no set-to-zero threshold was applied. The predicted interaction counts were scaled like the training inputs, using the same value ranges and thresholds.

For protein scaling, Hi-C matrices were first scaled to range [0...1000] with a set-to-zero threshold of 0.001. Value range scaling was then applied to the protein inputs in the same fashion as described above, using the same value ranges. For division-by-mean scaling, all proteins in a given dataset were divided by their respective mean without additional value range scaling.

Both for matrix- and protein scaling, bigwig files were used as inputs without dropping any values.

4.6 Concatenating datasets and predictions from different cell lines

To concatenate *predictions* from GM12878, HMEC, HUVEC and NHEK on K562, predictions from the first-named four cell lines on K562 were separately computed as before, using bigwig files without sample dropping and scaling as inputs. The four training matrices and the target matrix were scaled to [0...1000] with set-to-zero threshold of 0.001. Next, the four predicted cooler matrices were summed using `hicSumMatrices` version 3.4.1 and then divided by four using `hicNormalize` version 3.4.1 [46]. Details can be found in Listing 3 (p. 68).

To make a prediction on K562 from concatenated *datasets*, all of them, including the target dataset for K562, were first computed separately as usual, again taking bigwig files without sample dropping as protein inputs. Here, the matrices were scaled to [0...1000] with set-to-zero threshold 0.001 and the proteins were used in their original value ranges. The datasets from GM12878, HMEC, HUVEC and NHEK were then concatenated using the `concatTrainSet.py` python script [42] with the four datasets as inputs and standard options otherwise. Training was subsequently performed on the concatenated dataset, and finally the resulting model was taken to predict the target matrix as usual.

4.7 Emphasizing certain samples

To emphasize samples according to their interaction count, all samples were first given an equal weight of one. Next, samples with interaction counts in the range $[lower, upper]$ were selected, and their weight was adjusted such that the weight sum of the selected samples was k-times the weight sum of the unselected samples, Equation 2 and 3, where k is an adjustable numeric parameter.

$$k = \frac{targetWeight \cdot numberWeightedSamples}{1 \cdot numberUnweightedSamples} \tag{2}$$

$$targetWeight = k \cdot \frac{numberUnweightedSamples}{numberWeightedSamples} \tag{3}$$

The target weight was rounded to integer, since the default weight of the sklearn.ensemble methods is of integer type.

Finally, five-fold cross-validation (CV) was performed on the modified dataset, whereby the weights were only applied to the five training sets. To measure the performance of a certain parameter setting, the loss function was then defined by virtue of Equation 4

$$\text{loss}(lower, upper, k) = 1 - \text{mean}[\text{testScore}_1(lower, upper, k)...\text{testScore}_5(lower, upper, k)] \tag{4}$$

where testScore_n is defined as the R^2-coefficient of determination of the n-th CV-test set for the given parameters *lower*, *upper* and k. It holds that $\forall(lower, upper, k) \in \mathbb{R}^3$: $\text{loss} \in [0, \infty)$, since $R^2 \in (-\infty, 1]$ and thus $\forall n \in 1, 2, ..., 5 :\ \text{testScore}_n(lower, upper, k) \in (-\infty, 1]$.

To find a suitable factor and range for sample weighting, hyperopt [47] was applied, using its tree-parzen-estimator based search strategy to minimize the loss function. To this end, evaluations of the function were performed with 200 different parameter combinations, whereby *lower* and *upper* were uniformly sampled from the interaction count value range, and factor k was uniformly sampled from $[0.1, 100]$. Due to the long run time of the parameter search – 200 evaluations of the loss function took about 24 h on the available infrastructure – the optimized parameters could only be determined for chromosome 17 of the GM12878 cell line; these were subsequently applied to the full training set and used to predict interaction counts in K562.

For the CTCF-based sample-emphasizing, the same procedure as above was followed, only the sample selection was based on CTCF start-, end- and window-feature instead of interaction counts. The full code for searching interaction count- and CTCF-based sample weighting parameters can be found in the *weightingParameterSearch.py* module on github [42]. Among other adjustment parameters, the names of the features used for computing the weights can be modified from the command line, so that the code can also be used to compute weights based on other proteins than CTCF.

For the TAD-based sample weighting method, TADs were called on the training cell line using version 3.4.1 of `hicFindTADs`, Listing 4 (p. 69). Due to time constraints, again, only GM12878 was used for training, and the parameters were adjusted manually until the detected TADs "looked good" – unfortunately, there is no ground truth with regard to TADs, which is a clear weakness of this approach.

All samples within TADs shorter than 0.5 Mbp were then selected and weighted with factor k and $targetWeight$ defined as in Equation 3. Unlike interaction-count- and feature-based weighting, TAD-based weighting was only tested for fixed factors $k \in \{0.1, 1.0, 10, 100\}$, to save time but still find out if there was an effect worth being investigated in more detail. The length restriction was chosen on the one hand to emphasize structures shorter than 0.5 Mbp, which were often not present in the predictions, and on the other hand to exclude very long TADs which `hicFindTADs` often reports e. g. across centromere regions.

In order to remove the "diagonal" from the predictions as (probably) done by Zhang et al., all samples with $distance \leq 1$ were discarded, in this case both from the training and test datasets. Apart from that, predictions were computed as usual. Note that hicprediction records distance in bins, whereas HiC-Reg is using basepairs.

For all sample-weighting investigations, interaction counts were scaled to [0...1000], while protein data were taken in bigwig format, in their original value ranges and without dropping empty samples.

4.8 Comparison between HiC-Reg and hicprediction

To compare hicprediction to HiC-Reg, the corresponding input parameters were set to match, if known. This means that the default random forest was used for hicprediction (see 4.3), and empty samples were discarded, see subsections 4.4. The proteins were not scaled, but matrices were adjusted to a value range of [0...1000]. Hicprediction was then amended to allow a 5-fold cross-validation as described by Zhang et al. Here, the training data was split using sklearn's `cross_validate()` function, five models were trained and subsequently used for predicting the target cell line. The single predictions were then combined into one by taking the mean analogous to subsection 4.6.

To generate high-quality plots of the predictions in [2], supplementary data were downloaded from [48, 49, 50] and converted to cooler format using hicprediction's *convertHiCRegPrediction.py* script [42]. That same script also computes Pearson correlation with respect to the target interaction counts stored in the data. It is not exactly known how the target data has been obtained, but it must have been processed in some way, as the value range differs from the original one in [13].

To reduce the impact of different input data and improve comparability, HiC-Reg predictions were additionally computed with training data generated by hicprediction. For this purpose, five hicprediction (cross-validation) training- and test datasets were generated as described above and then converted to text format using the *convertForHicReg.py* script to be found on github [42]. With these datasets, training and prediction were performed using the `regForest` binary as described in sections 2.1 and 2.2 of the HiC-Reg readme [24]. The numeric parameters leaf size l, maxfactorsize k and number of trees n were chosen as in the readme, i. e. $k = 1$, $l = 10$, $n = 20$ and the priors file (`-b` option) is created by the *convertForHicReg.py* script, too. Since the `regForest` binary was not available from github, it was recompiled with g++ version 5.4.0 (20160609) on Ubuntu 16.04. LTS, dynamically linking against the corresponding libgslcblas- and libgsl libraries. Due to technical problems with our computation cluster and HiC-Reg's memory demand

at higher resolutions – around 120 GB for a resolution of 5 kbp – the comparisons described above could only be computed at 25 kbp resolution.

Additionally, WINDOW-type predictions were made with HiC-Reg from ground up, taking the same Hi-C matrices and BAM files[2] as inputs that were used for hicprediction, cf. subsection 4.1 and 4.2. To this end, the BAM files were split up per chromosome using `samtools` version 1.9 and the coverage was computed using `bedtools genomecov` v2.29.2, see Listing 5 for an example. This produces a bedgraph-formatted output which can serve as input for binning. The latter was then performed with HiC-Reg's `aggregateSignal` binary, taking the bedgraph files as inputs, while resolution was set to 5 kbp and the hg19 reference genome was used. Unfortunately, the `aggregateSignal` binary requires a third, yet underspecified input, the so-called region file, which was just copied from the examples provided in the repository. Next, Hi-C matrices were converted to text using the *convertForHicReg.py* script again, this time discarding its other outputs. Window features for HiC-Reg were then generated for all 12 proteins, using the `genDatasetsRH` binary as described in the readme [24]; refer to Listing 6 for an example.

Both the aggregateSignal- and genDatasetsRH binaries are present in the repository, but were also recompiled due to problems with dynamic linking.

Due to time constraints, comparisons between hicprediction and HiC-Reg were only made for cross-cell predictions GM12878 on K562, chromosome 17, using the WINDOW method.

Ressource consumption for all steps above was measured with GNU time 1.7 on a machine equipped with a quad Intel® Core™ i5-4200U CPU at 1.6 GHz and 8 GB memory, refer to Listing 7 for the parameters used.

[2] here, replicate 2 only; to be repeated with both replicates

5 Results

In the following subsections, the results for the changes to hicprediction introduced in section 3 will be presented and interpreted. Furthermore, subsection 5.7 provides a direct comparison of the (improved) results from hicprediction to the ones from HiC-Reg.

5.1 Avoiding training samples without protein data

The effect of removing empty samples from the datasets was clearly visible, both in terms of Pearson correlation and matrix plots, see Figure 14 and 15. It is obvious that inverted triangles and gradient-style predictions were removed by discarding empty samples, but the prediction is also strongly fragmented, so that structures like TADs and loops are hardly visible in the plots. For example, in GM12878 and K562, only about 25% of the originally 3 227 900 samples remained in the datasets, which means that 75% of all pairs in the predicted Hi-C matrix for K562 had the default interaction count zero. This effect would be even more striking for the explanation example from subsection 3.1, where only 6 of 79 900 samples would remain in the dataset. In this (staged) case, the algorithm then just "learns" the relation "sample exists ⇒ read count is 100", without any of the features (start, end, window, distance) being considered as a decision criterion.

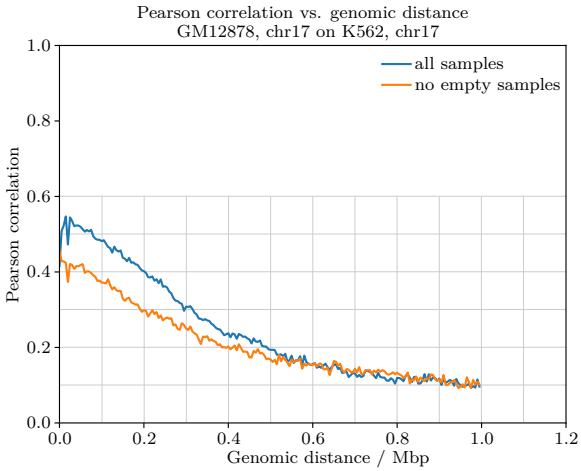


Figure 14: Pearson correlation before/after removing samples

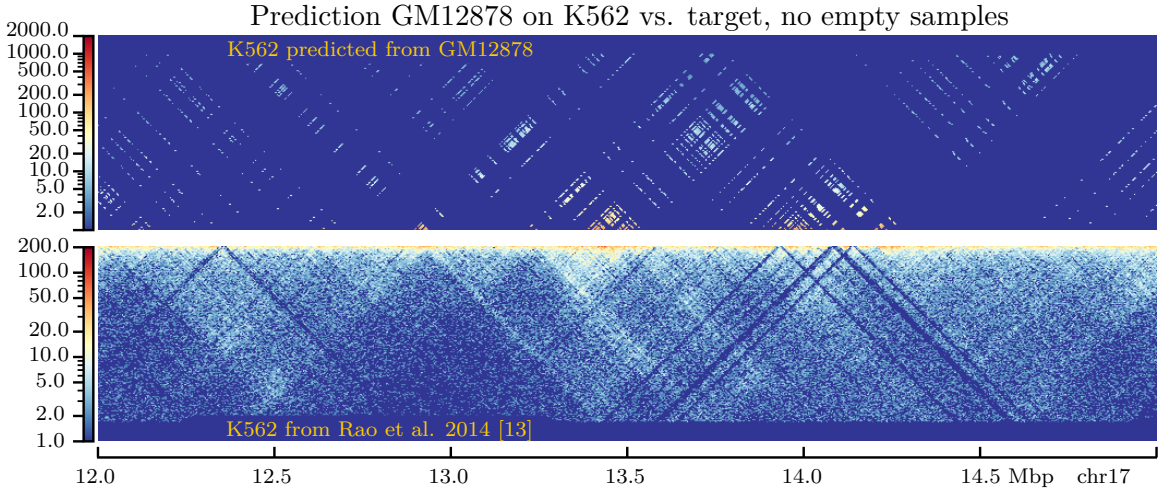


Figure 15: Example prediction, without empty samples

5.2 Dealing with sparse input data

Filtering fragmented output matrices like the ones obtained in subsection 5.1 was unsuccessful, since no parametrization of the 2D-Gaussian filters proved useful – either the gaps were not filled, or the structures in the matrix were blurred too much, or both, see Figure 16. Other kernel types and more sophisticated image-inpainting methods [51] might do better, but choosing the appropriate ones and adapting them to the problem at hand would be laborious and beyond the scope of this masterproject. Additionally, and even more important, it is not clear whether such sparse matrices as depicted in Figure 15 would be a good starting point for suchlike approaches. The missing regions can be several 100 000s of base pairs wide (e. g. Figure 15, upper right), and both the low Pearson correlation (Figure 14) and the lack of structure in the predicted matrices (Figure 15) suggest there might simply not be enough information for imputing missing values, with whatever approach.

Filtering the inputs with one-dimensional Gaussian filters was more successful. While it seems impossible to fill long stretches of empty bins by Gaussian smoothing alone, some structures became quite well visible and the Pearson correlation also improved slightly, Figure 17 and 18. Apart from the gaps remaining in the prediction, smoothing the proteins also has the disadvantage of blurring and shifting the structures in the predictions compared to the target matrices. This is typical for Gauss filters, but probably acceptable in this case.

In predictions from bigwig files, there were much less empty bins and no inverted triangles or gradient-style regions, Figure 19 and Figure 40 to 42 on page 51 ff. Due to the higher peak density in the bigwig files, the effect of leaving out empty samples was sometimes hardly recognizable, compare Figure 19 and 20. The Pearson correlations were also similar for both approaches, and both were better than the ones from using peak files, Figure 21. However, the effect of removing samples was found to be dependent on cell line, chromosome and genomic position. Some chromosomes, e. g. 21, have large contiguous regions which seem to be deprived of all proteins, probably for biological reasons. For such regions, removing empty samples was significantly better, Figure 43, but detrimental for other regions of the same chromosome, see Figure 44. This was unfortunately not reflected in the Pearson correlation, Figure 47. Note that the area under the correlation curve is up to 0.7, which is a surprisingly high value considering the modest look of the matrix plots, Figure 43 and 44.

With regard to background signal and sequencing noise in the bigwig files, the results from chromosome 17 suggest that the algorithm can at least partially learn which ChIP-seq read count values are indicative of high interaction counts and which are not. Beyond that, using a threshold to set signal values below a certain level to zero is possible, but finding a value which works well across all cell lines, chromosomes and proteins seems challenging. Too large values again lead to sparsity and too small values do not have noteworthy influence on the result.

Avoiding empty bins in the inputs by increasing the input matrix resolution (bin size) to 25 kbp worked surprisingly well, even without further measures, Figure 22. Especially larger structures like the ones from 12...13 Mbp and 13...14 Mbp were usually at least signified. However, gradient-style regions and inverted triangles triangles still occurred,

albeit to a lesser extent, since the underlying problem remains the same, see e. g. the regions around 13 and 14.5 Mbp in Figure 22.

As expected, discarding empty samples led to less fragmentation with 25 kbp than with 5 kbp, because fewer input bins were empty; the result was still not useful, Figure 23. Again, it was not possible to fill in missing values with a Gauss filter without blurring the matrix too much, Figure 24. For 25 kbp resolution, too, smoothing the proteins worked better than smoothing the predicted matrix. Larger gaps could again not be closed, but apart from that, structures were often more easy to identify than without smoothing, compare Figure 22 and 25.

Predictions using bigwig files were also performed, but at 25 kbp resolution, it was even more difficult to say whether these were better or worse than the ones from peak files, compare Figure 28 and 22. As already mentioned above, it cannot generally be said whether leaving out or keeping empty samples is better. In the example matrix snippet from chromosome 17, it was again difficult to determine differences, Figure 28, 29 and 30, but this might not hold for all regions.

In terms of Pearson correlation, all predictions from 25 kbp resolution were better than the best one from 5 kbp, Figure 30, and structures indeed sometimes seemed more recognizable in the plots for 25 kbp. Interestingly, the fragmented prediction from peak files without empty samples, Figure 23, showed one of the best Pearson correlations obtained thus far. This is probably because interaction counts of discarded regions are set to zero, which seems to be correct in many cases, especially in the distance range [0.4...1.0] Mbp. Independent of the root cause, this example again underscores that relying on Pearson correlation alone is unfavorable, because the corresponding matrices can be useless in practice.

For 25 kbp, predictions were also performed using bigwig- and peak files simultaneously, thus increasing the number of features in the samples from $3 \cdot 12 + 1 = 37$ to $3 \cdot 24 + 1 = 73$ (2x12 start-, 2x12 window- and 2x12 end features plus distance). While the Pearson correlations were mostly better than with bigwig files alone, especially when keeping empty samples, Figure 48a and 48b, not much improvement was visible in the predicted matrices, Figure 31, 45 and 46. Feature importance plots show that the influence of the start- and end features from the peak files was generally negligible, but the window features were often considered more important than most features from bigwig files, Figure 26 and 27.

5.3 Scaling protein signal values and Hi-C interaction counts

For three of four tested scales for Hi-C interaction counts, [0..10], [0...100] and [0...1000], predicted and (scaled) target value were in the desired order of magnitude and the Pearson correlation was very similar to the status quo, see Figure 32a and 49. For value range [0...1], the Pearson correlation was notably worse than without scaling, at least for distances greater than 0.2 Mbp. No obvious reason for this could be found in the plots of the predicted matrices, but the usual "logarithm plus 1"-representation is also not well suited for matrices with interaction counts generally smaller than 1. The target matrices, too,

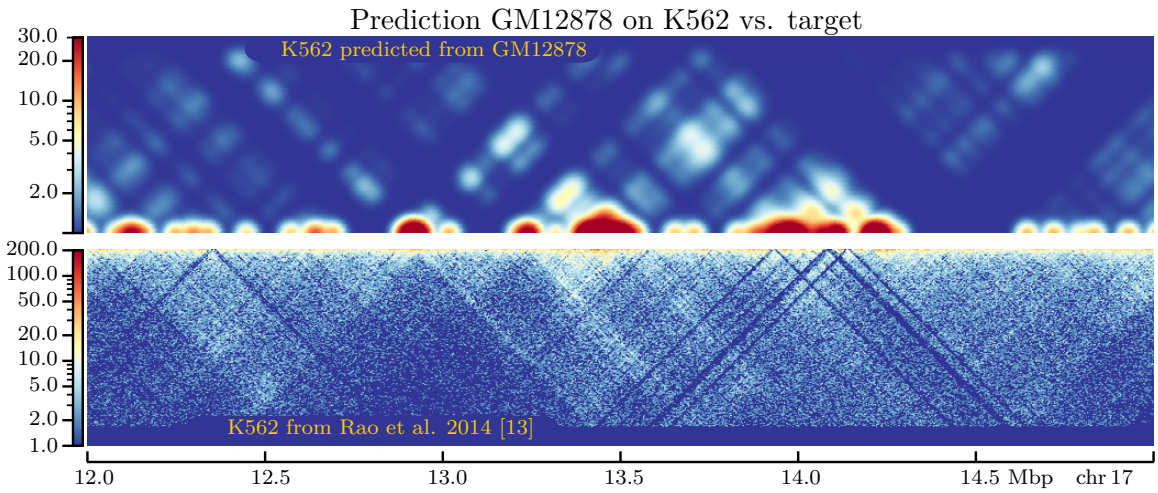


Figure 16: Example prediction from peaks, no empty samples, 5 kbp, matrix filtered $\sigma = 5.0$

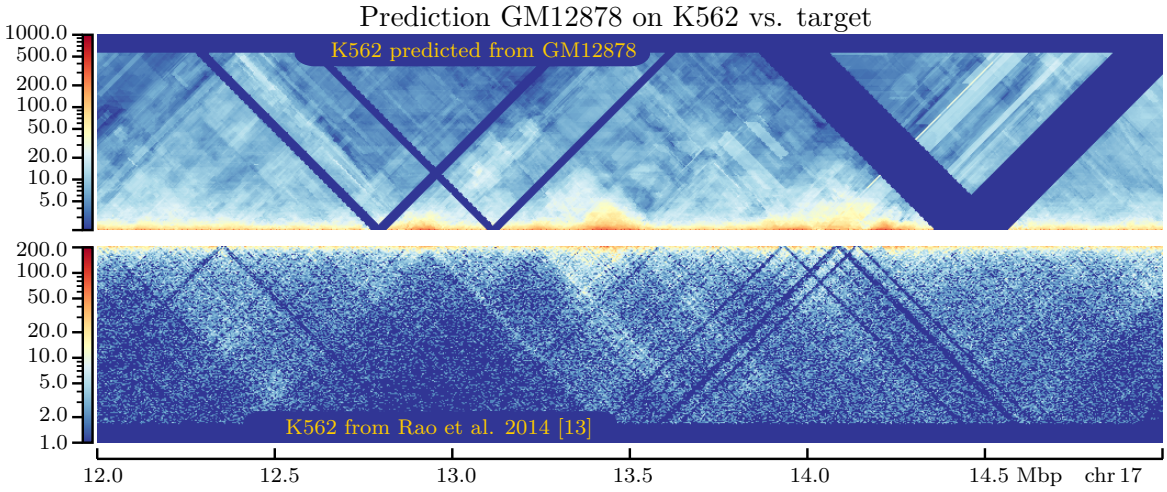


Figure 17: Example prediction from peaks, no empty samples, 5 kbp, proteins filtered $\sigma = 4.0$

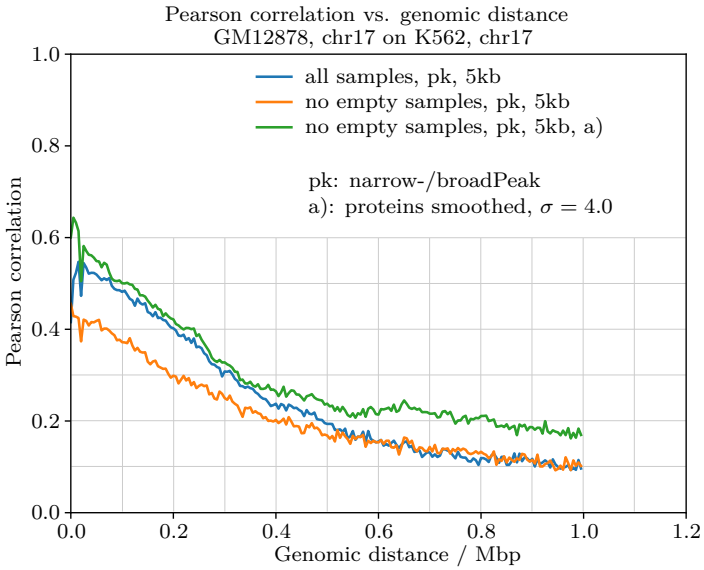


Figure 18: Pearson correlation, 5 kbp, proteins filtered $\sigma = 4.0$

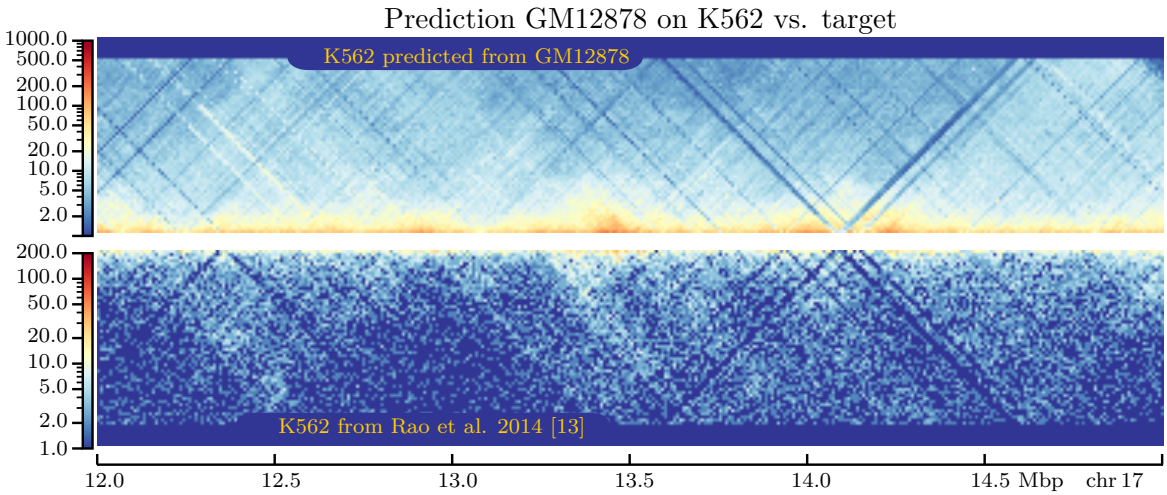


Figure 19: Example prediction from bigwig, all samples

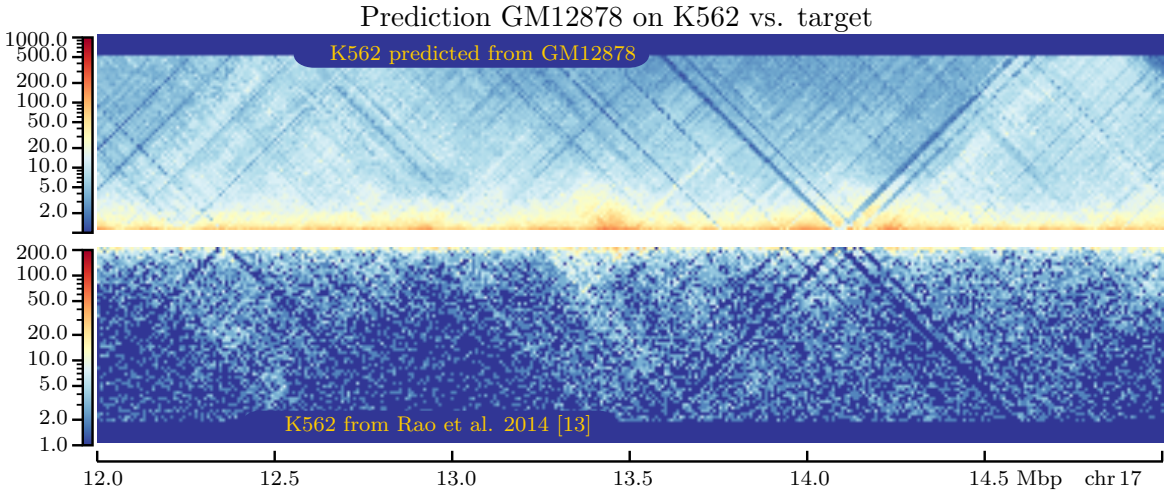


Figure 20: Example prediction from bigwig, no empty samples

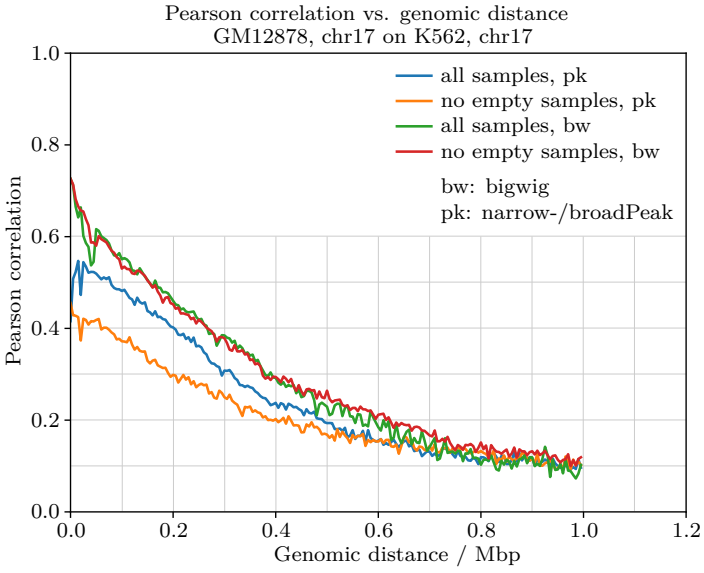


Figure 21: Pearson correlation before/after removing samples, 5 kbp

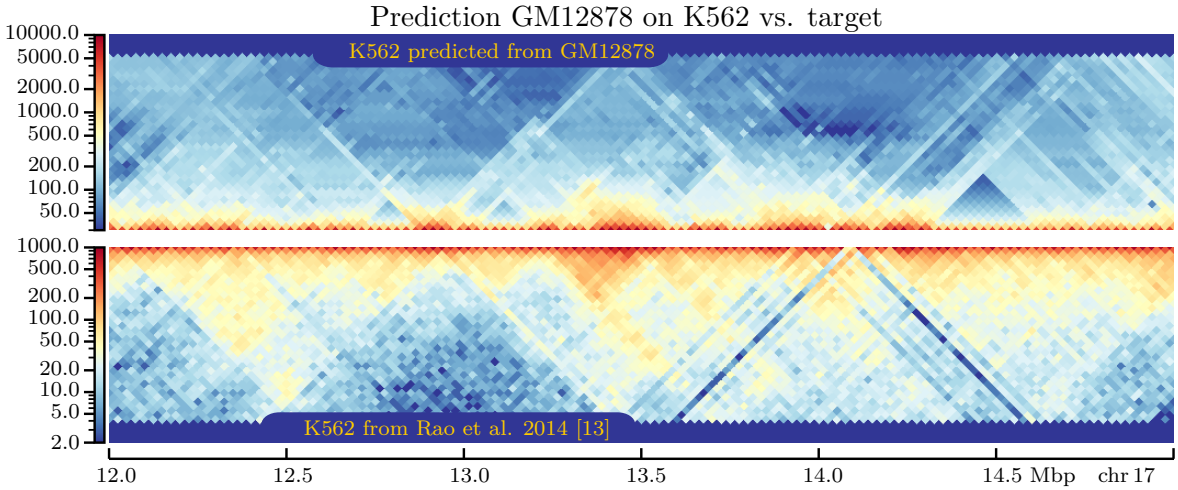


Figure 22: Example prediction from narrow-/broadPeak, 25 kbp, all samples

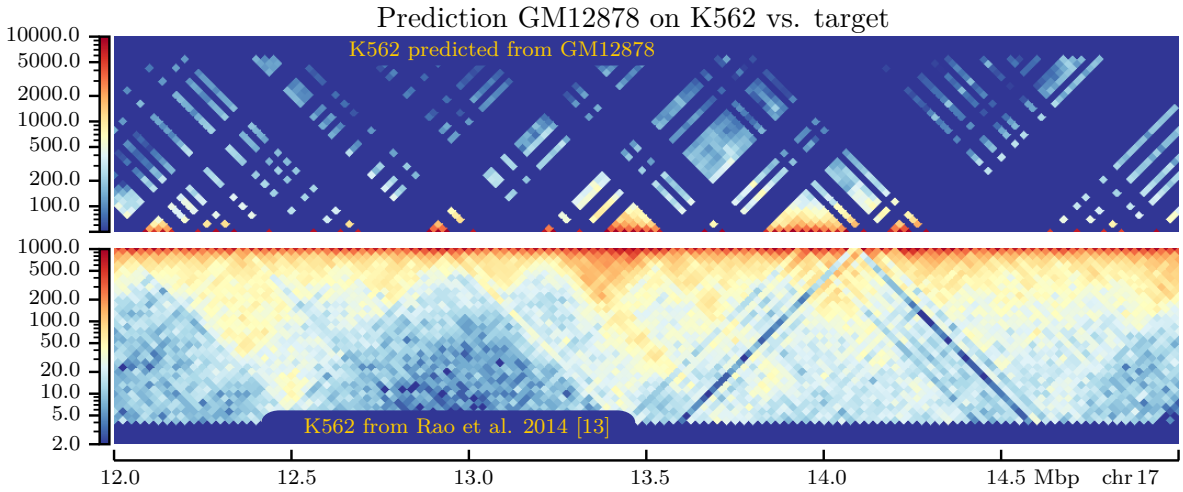


Figure 23: Example prediction from peaks, no empty samples, 25 kbp

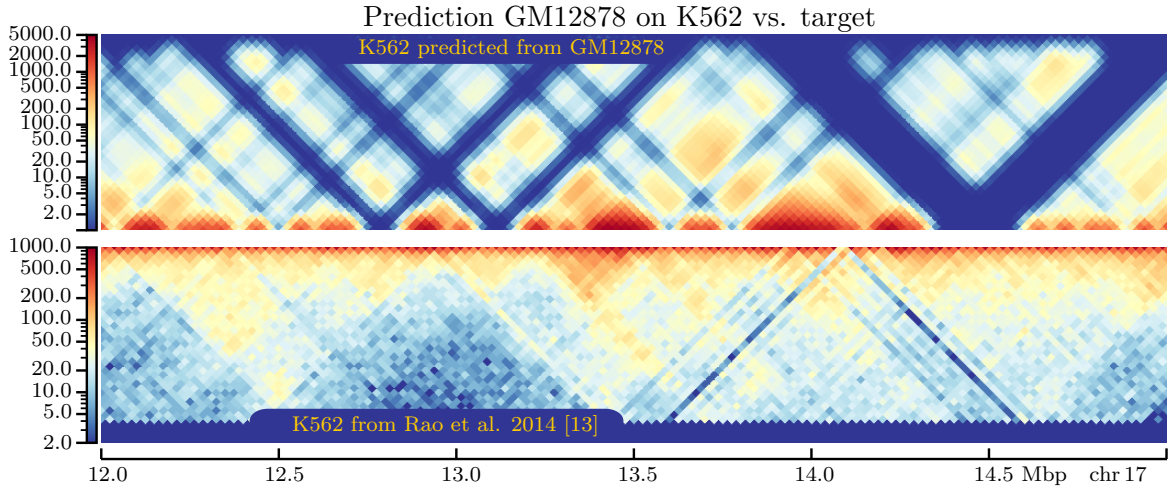


Figure 24: Example prediction from peaks, no empty samples, 25 kbp, matrix filtered $\sigma = 1.0$

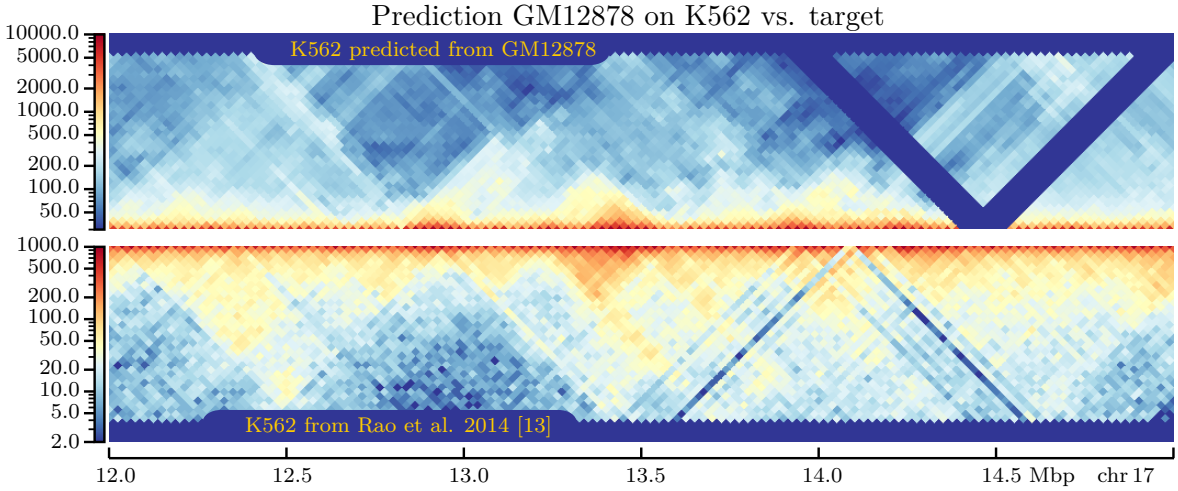


Figure 25: Example prediction from peaks, no empty samples, 25 kbp, proteins filtered $\sigma = 1.0$

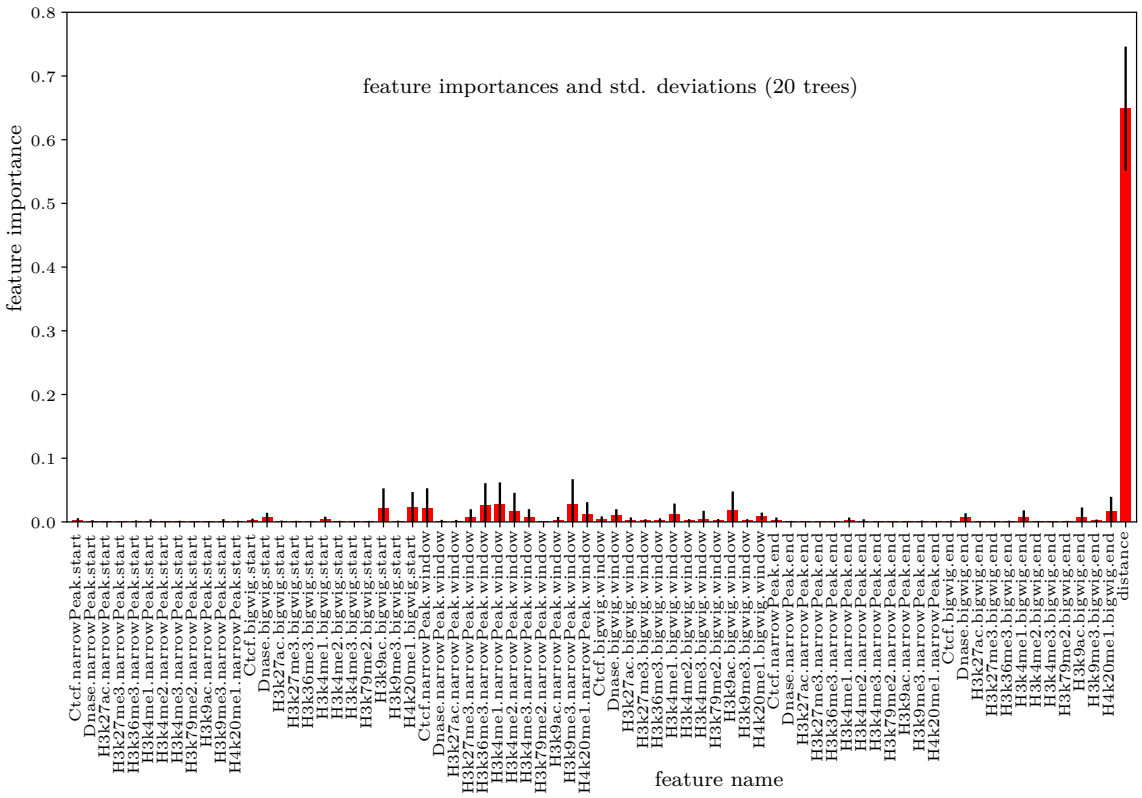


Figure 26: feature importances, GM12878 on K562, chr17, bigwig- and peak files combined

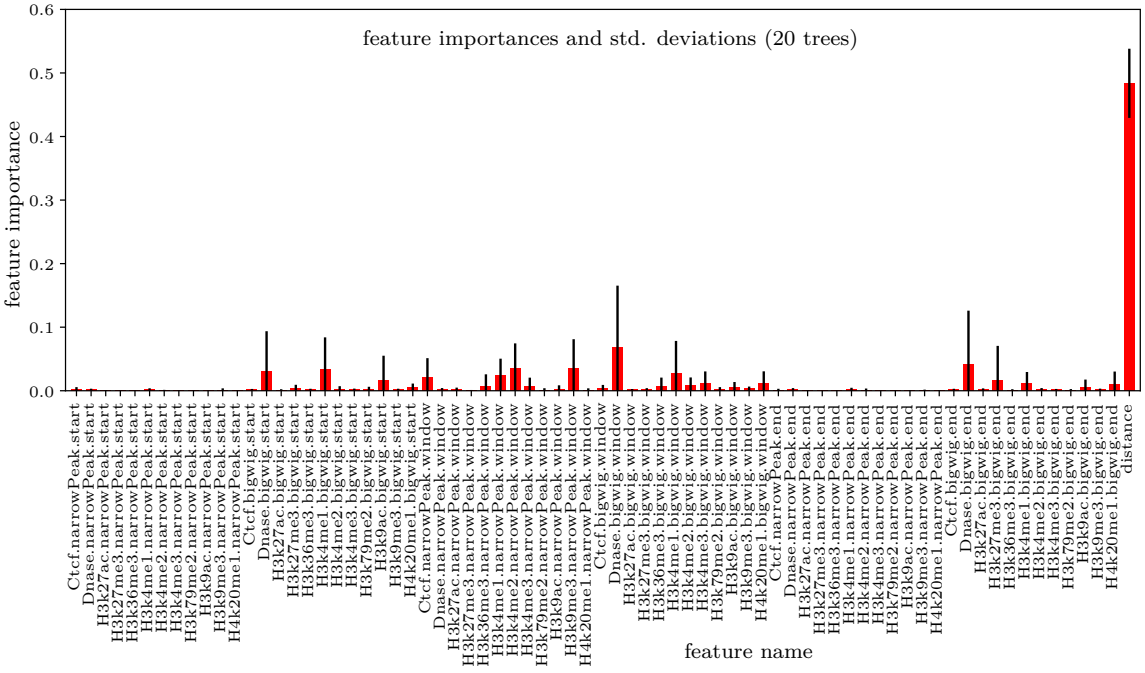


Figure 27: feature importances, GM12878 on K562, chr21, bigwig- and peak files combined

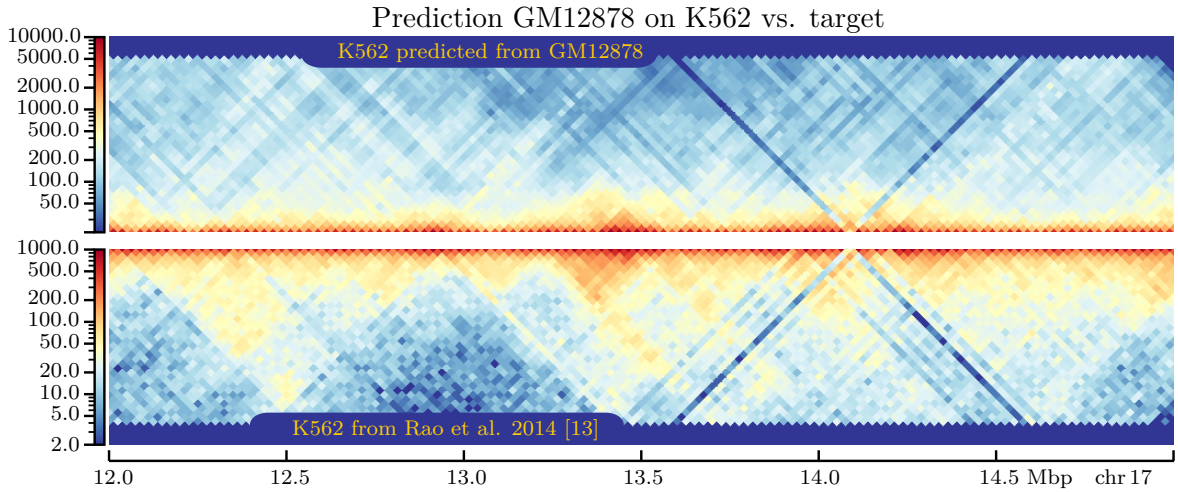


Figure 28: Example prediction from bigwig, 25 kbp, all samples

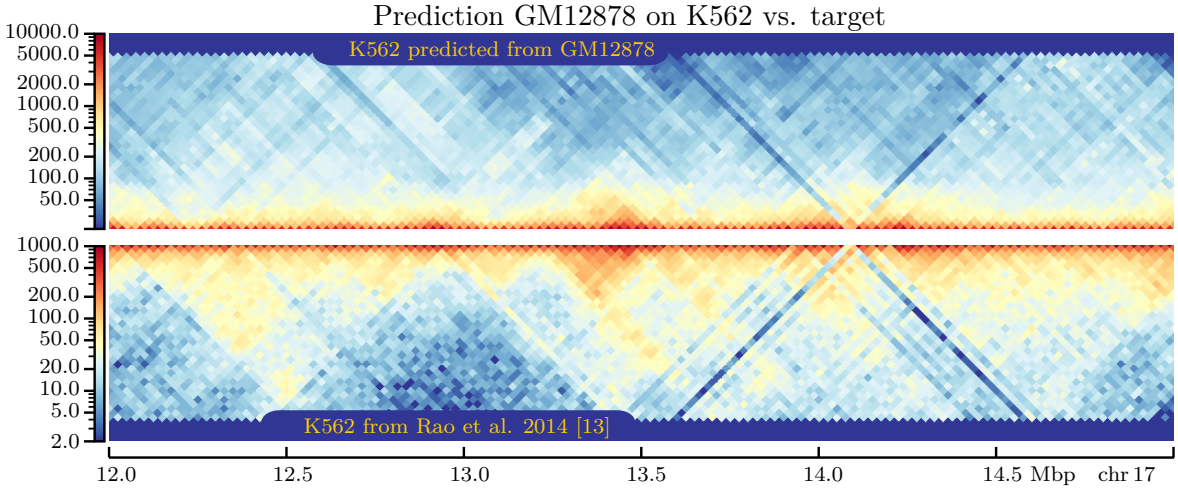


Figure 29: Example prediction from bigwig, no empty samples, 25 kbp

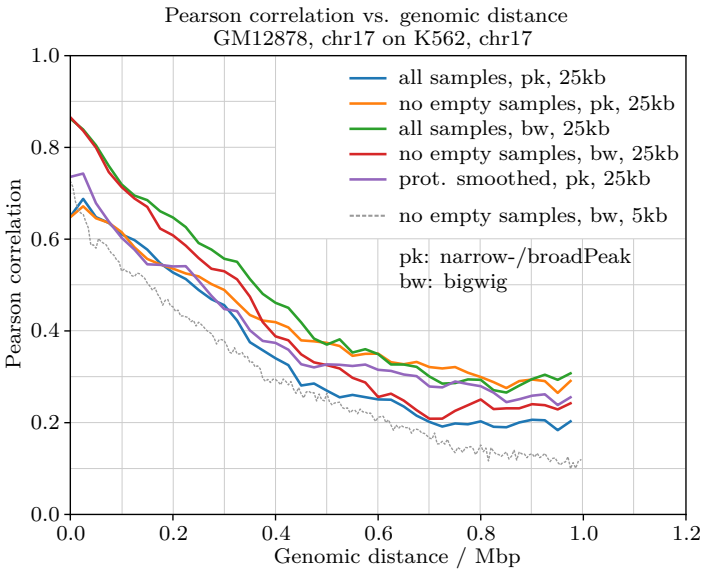


Figure 30: Pearson correlation comparison

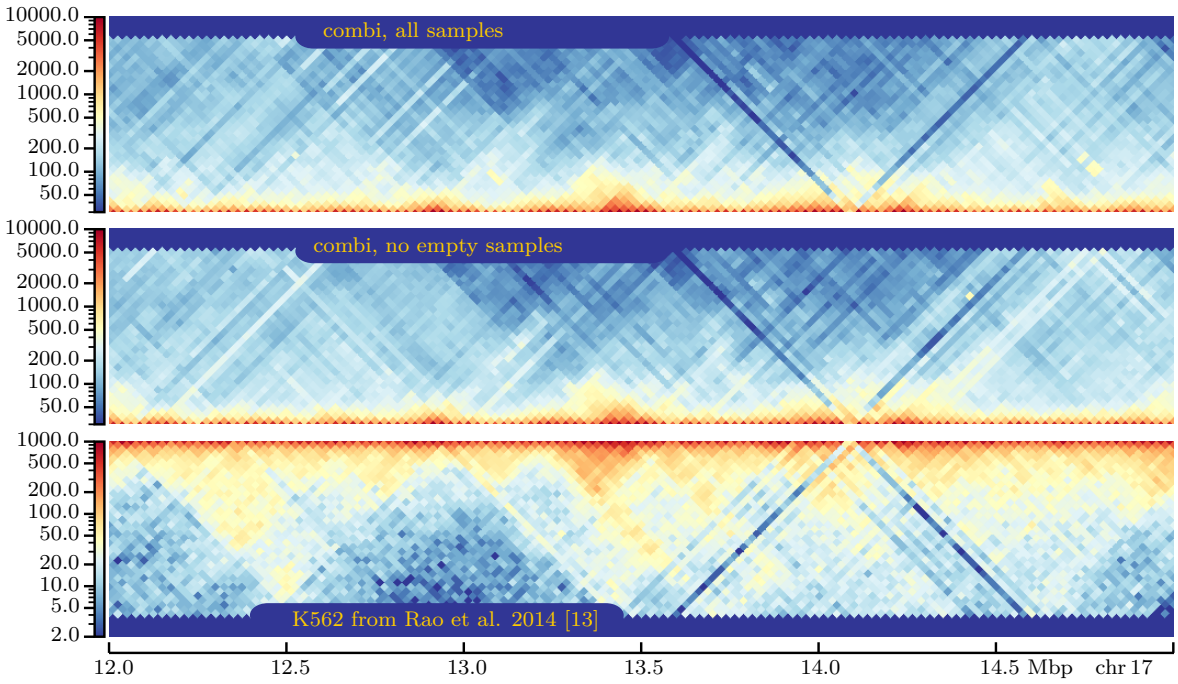
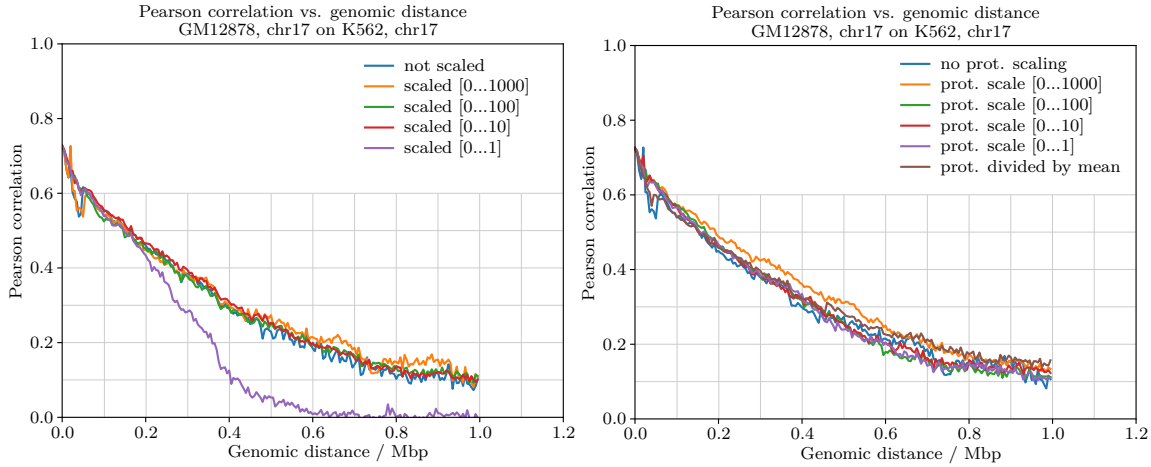


Figure 31: Example combined prediction, 25 kbp

do not look very informative when scaled and plotted this way, which is a reason on its own to go without the [0...1] range.



(a) matrices scaled (b) proteins scaled

Figure 32: Pearson correlations, matrices / proteins scaled

Feature scaling yielded the expected outcome in terms of Pearson correlations, Figure 32b, i. e. all value ranges had similar performance and were no better or worse than without. In terms of Hi-C matrices, the results were surprisingly worse than without scaling, with errors in the predictions e. g. around 13.5 and 14.0 Mbp, see Figure 50.

The comparatively bad outcome might partially be due to the small set-to-zero threshold which was applied after adjusting the value range, subsection 4.5. As this is a non-continuous transformation of the value range, thresholding could in principle affect the splitting. However, it would be very surprising if this was the main cause, as only very few values fell below the threshold for each protein; for example, in value range [0...10], at most 53, or 0.3% of 16 240 samples from a single protein were set to zero. Additionally, the threshold hypothesis does not explain the errors in predictions with value range [0...1], where the threshold was not applied. On the other hand, it is also hard to imagine that the errors in the predictions occurred by chance, as they were approximately at the same positions across all concerned predictions.

To investigate further on the issue, predictions were also made for 25 kbp resolution, Here, too, the outcome was partially unexpected. While the Pearson correlations indicated a positive influence of scaling for all but the division-by-mean method, Figure 54, the plots of the matrices looked very similar to the non-scaled state, Figure 51. The reason for this behavior could unfortunately not be determined.

Because scaling input features does not seem very common for random forests, only a single paper was found on the topic. Here, Dinc. et al. investigated the influence of value-range (min-max) scaling and z-score normalization[3] on the performance of several classifiers for protein crystallization images [52]. For the random forest classifier, some improvements for the z-score normalization, but only very little change for value-range scaling were recorded.

[3] subtract mean and divide by standard deviation

However, the reasons for the improvement were not investigated. The already mentioned study by Strobl et colleagues [34] does not cover feature scaling directly, but found that features with different value ranges can cause bias in variable selection, especially when used in combination with categorical variables and bootstrapping with replacement.

Within this masterproject, it could not be clarified what exactly caused the unexpected results when scaling feature value ranges. However, first tests showed an interesting relation between the floating point precision of the feature values and the predicted matrices, which could be investigated in future studies. Currently, hicprediction is using 32-bit floating point numbers rounded to 6 digits after the decimal point – and this may be insufficient for small value ranges, considering the strongly nonuniform interaction count distribution, Figure 12. For example, when using value range [0...1] and 5 kbp matrix resolution, there are 100 000 different floating point numbers in the interval [0, 0.100000) after rounding to six digits after the decimal point – but more than half of the approximately 3.2 million samples have interaction counts lying in this interval.

Since protein scaling yielded unfavorable results, the remaining computations were performed without.

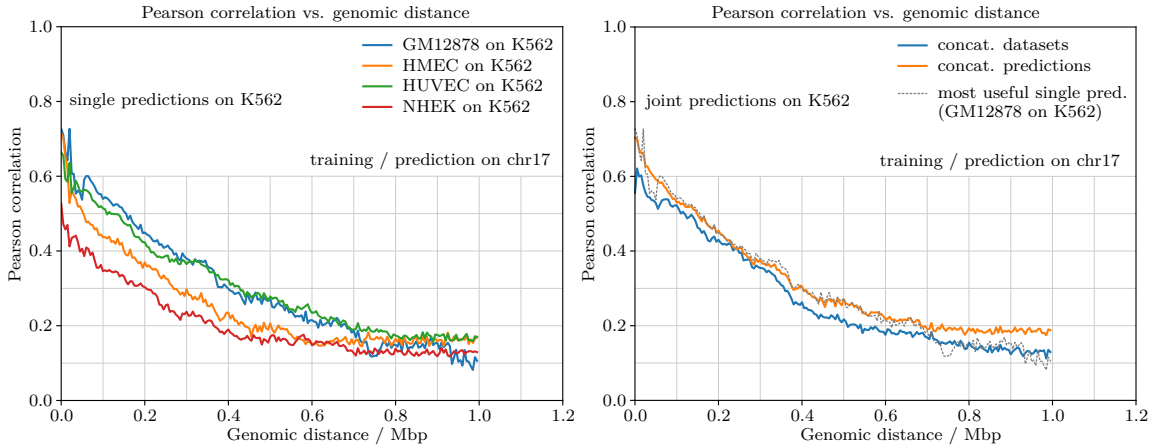
5.4 Concatenating datasets and predictions from different cell lines

In order to combine predictions from different cell lines on K562, the single predictions were computed first, Figure 55, top 4 panels. It is obvious that not all cell lines are equally well suited for predicting K562, which is probably due to different biological functions. Although e. g. the prediction from HUVEC showed only fairly few structures, the averaged prediction from GM12878, HMEC, HUVEC and NHEK on K562 was still acceptable, partially even better than the best single-cell-line prediction, Figure 33 and 55. Both the Pearson correlation and the plot of the predicted matrix look more smooth than the ones from single cell lines, which is due to the averaging process and thus not surprising. The structures in the matrix plot were still distinguishable and not significantly worse than the ones from single-cell-line predictions.

The prediction from concatenated datasets performed not as well as the averaged single predictions, but still better than the single predictions from "less suitable" cell lines. It should however be noted that the datasets have been concatenated without prior feature scaling, due to the problems mentioned in subsection 5.3 above. This is probably suboptimal, because the same features can have significantly different value ranges in different cell lines, see Figure 52a. The predictions from concatenated datasets might thus improve once the existing feature scaling problems have been investigated and resolved.

5.5 Emphasizing certain samples

After 200 runs of the tree-structured Parzen estimator, see subsection 4.7, the supposedly best parameters for weighting samples according to interaction count were determined as $lower = 292.635$, $upper = 857.073$ and $k = 14.625$. This means that all samples with an interaction count in the range [292.635...857.073] were given integer weights such that the sum of the weighted samples was approximately 14.625 times the weight sum (=number)



(a) single predictions (b) joint predictions

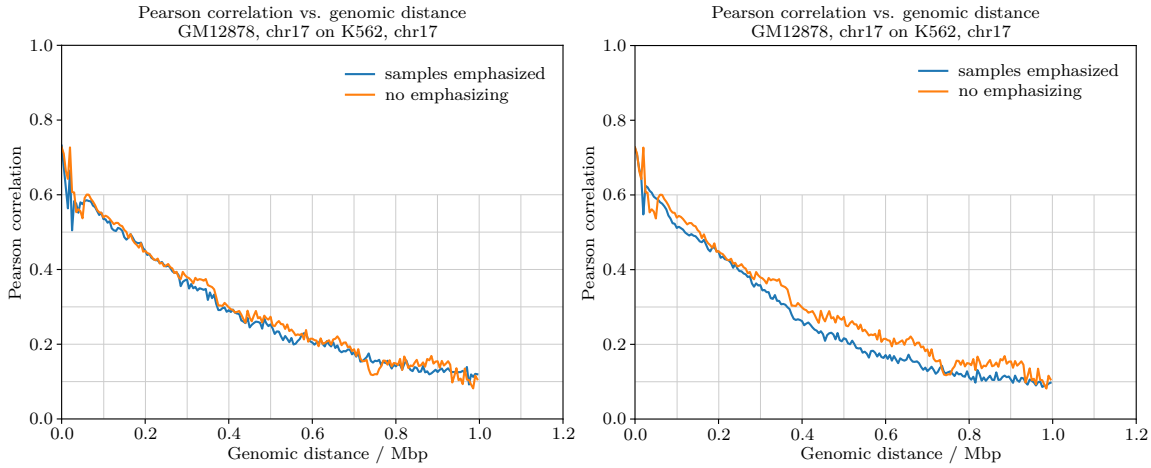
Figure 33: Pearson correlations, single vs. joint predictions on K562

of the unweighted samples. Unfortunately, there were only six samples in the given range, so the effect of interaction-count-based weighting on the predictions was quite small. Both the resulting Pearson correlation and the matrix plots looked fairly similar for predictions with weighted samples and usual predictions, Figure 34a and 57.

With another 200 runs of the estimator, the supposedly best parameters for weighting samples according to CTCF signal value were determined as $lower = 228.8$, $upper = 670.112$ and $k = 4.241$. In this case, there were 285 242 samples with CTCF start-, end- or window-feature in the given range, so an influence of CTCF-based weighting on the results was to be expected, which also precipitated in the feature importances, Figure 59. Interestingly, only the CTCF-window feature gained importance, while start- and end-feature remained at low levels. However, the Pearson correlation of the prediction from weighted samples was actually worse than before, Figure 34b, and the matrix plots showed at least no significant improvement, fig 58.

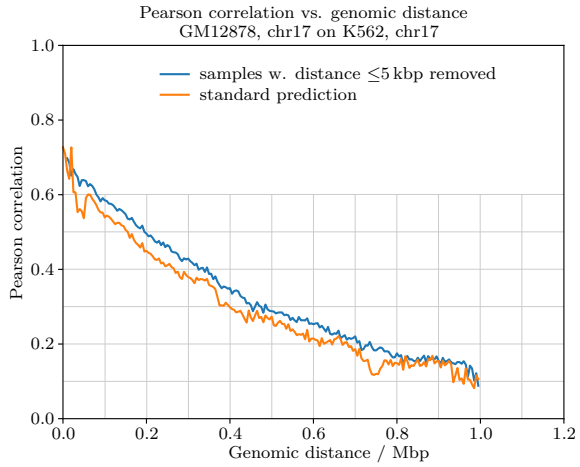
Removing all samples with a distance below 5000 bp surprisingly improved and also slightly smoothened the Pearson correlation, Figure 34c. The matrix plots also clearly differed from the standard predictions, but it is difficult to say whether they are better, Figure 60. While some predicted structures were more distinct, there were also some which do not seem to match any real interacting regions, and the contrast between interacting- and non-interacting regions also seemed lower than in the standard predictions.

TAD-based weighting of samples did not change the prediction results much. Both the Pearson correlation and the resulting matrices were quite similar to the ones from standard predictions, Figure 34d and 61. Note that for weighting factor $k = 0.1$ the results were identical to the the status quo, because it turned out that $\frac{\sum_{weightedSamples} weight}{\sum_{unweightedSamples} weight} \approx 0.1$, so no weighting was performed at all due to rounding (subsection 4.7).

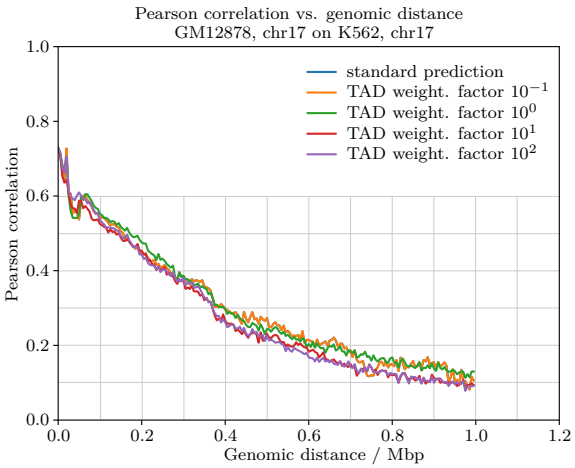


(a) weighting according to interaction counts

(b) weighting according to CTCF signal value



(c) removing samples with distance $\leq 5\,\mathrm{kbp}$



(d) weighting according to TADs

Figure 34: Pearson correlations for different sample weighting approaches

5.6 Replacing random forest by extra tree regression

Predictions using the extra trees algorithm were generally similar to predictions from random forests. While Pearson correlations were in favor of the random forest algorithm, Figure 35, the matrix plots looked fairly similar, Figure 56. It remains for future research whether this holds for other cell lines and chromosomes as well.

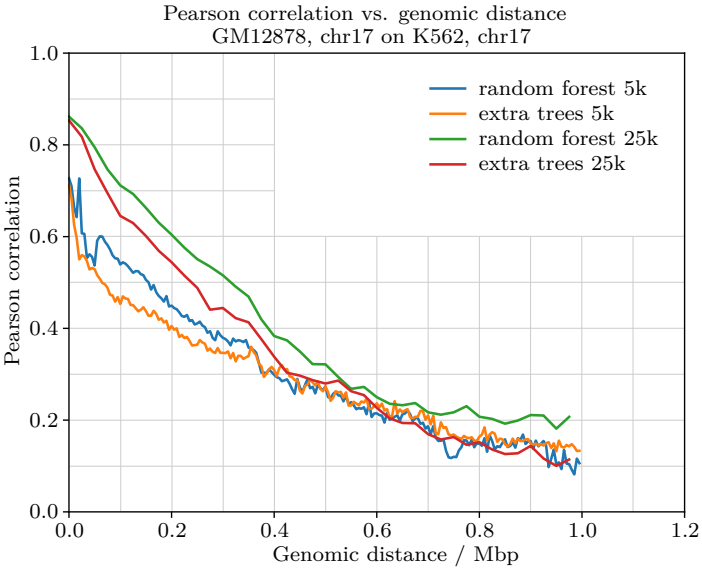


Figure 35: Pearson correlations extra trees vs. random forest

5.7 Comparison between HiC-Reg and hicprediction

Looking at the matrices obtained with hicprediction thus far, it was obvious that – despite some improvements compared to the original state – the predictions were still inferior to the ones published for HiC-Reg. Figure 36 shows a direct juxtaposition between a cross-cell prediction from HiC-Reg – reconstructed from data published along with the article [2] – and the corresponding prediction from hicprediction. In terms of Pearson correlation, the results from hicprediction and HiC-Reg were very different, see Figure 39a. While hicprediction was sometimes better at smaller distances, in the given test case particularly obvious below 0.5 Mbp, the published results from HiC-Reg were always – not only in the test case plotted in Figure 39a – better for larger distances.

To check whether the comparatively worse results from hicprediction where due to the underlying algorithm or to the input data, new HiC-Reg predictions were made by converting the corresponding hicprediction training sets to text files and using them for training and prediction with HiC-Reg. It is obvious from Figure 37 that the results did no longer differ much, except for the value range. This indeed points to the input data as the main cause for the differences between hicprediction and HiC-Reg.

However, the computations from hicprediction were much more efficient both in terms of runtime and memory consumption, see Table 6. While training took almost three hours

in HiC-Reg, it finished within around 90 seconds in hicprediction – and this even includes the time spent to split the data into five cross-validation sets and store them. No efforts were made to find the cause for the significant runtime difference, but it was obvious that HiC-Reg was using only one CPU, while hicprediction was set up to use multithreading on all available (four) CPUs for the random forest regressor.

	training		**prediction**	
	runtime / min	memory / GB	runtime / min	memory / GB
HiC-Reg	175,58	4,29	3,70	1,92
hicprediction	1,26	0,70	1,18	1,17

Table 6: computational effort HiC-Reg vs. hicprediction

For the example of CTCF in GM12878, the bedgraph input file converted from the corresponding BAM file proved qualitatively quite similar to the example file in the HiC-Reg github repository [24][4], see Figure 62 and 63. However, the newly created file contained about 55 000 (14.2%) more lines than the example file, which suggests that the original input data to HiC-Reg must have been filtered in some way. The paper [2] provides general information on how the data have been processed, e. g. with regard to software tools, but unfortunately lacks detailed information for example on specific filtering procedures or parameters. Irrespective of this, predictions with HiC-Reg using the converted BAM files did not yield useful results, see Figure 38. It could not be clarified what went wrong, but gradient predictions like these can occur when only distance is considered as a feature, cf. subsection 3.1. This could have happened either due to a bug in HiC-Reg or a misunderstanding of its input file formats, some of which are only specified by example.

A concluding discussion of all changes made to hicprediction within this masterproject will be given in section 6.

[4] full url: `https://github.com/Roy-lab/HiC-Reg/blob/master/Scripts/aggregateSignalInRegion/wgEncodeBroadHistoneGm12878CtcfStdRawDataRep1_chr17.counts`

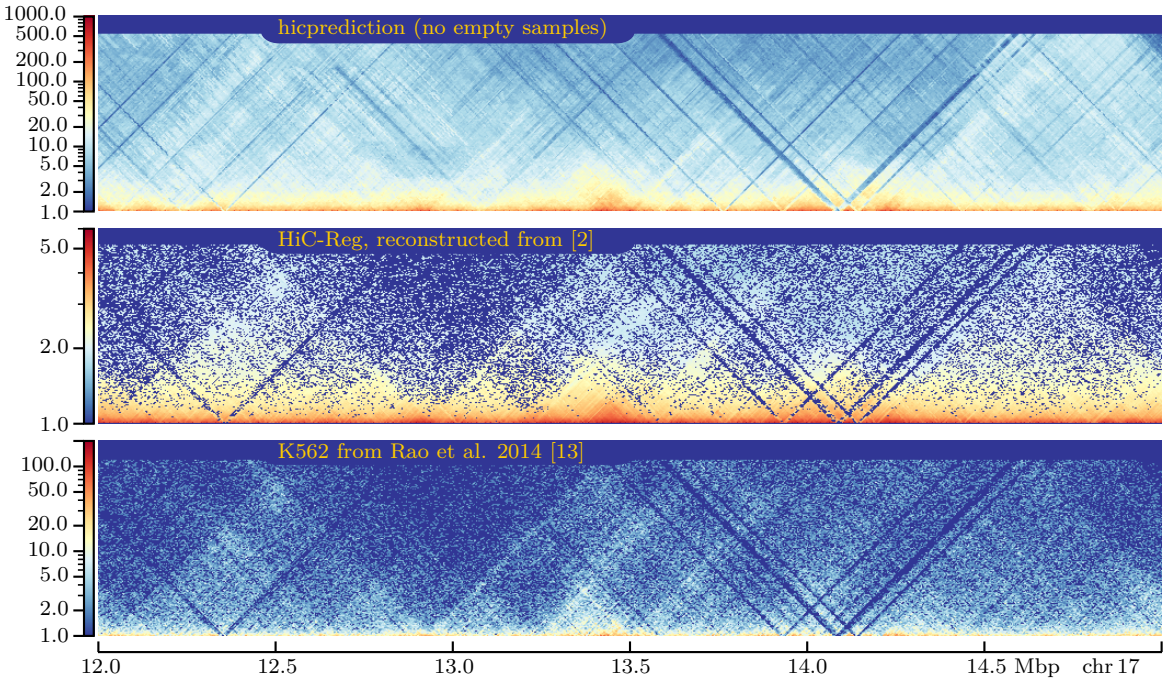


Figure 36: HiC-Reg vs. hicprediction, GM12878 on K562, chromosome 17, 5 kbp

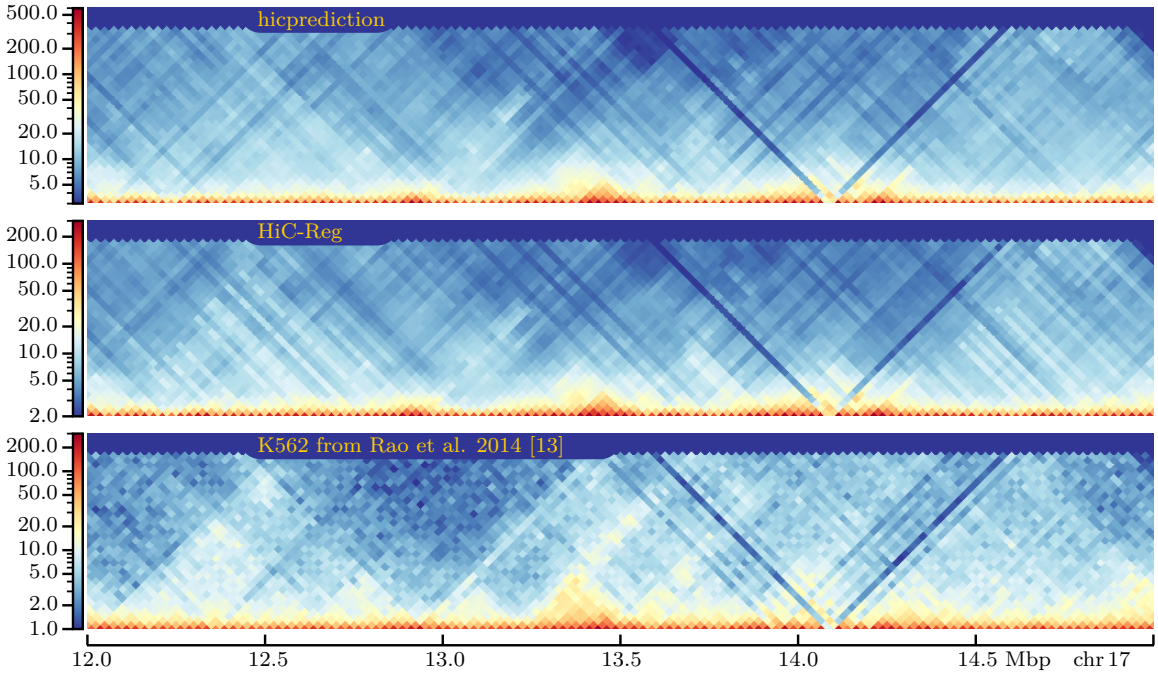


Figure 37: HiC-Reg vs. hicprediction, GM12878 on K562, chromosome 17, 25 kbp, HiC-Reg input data converted from hicprediction datasets

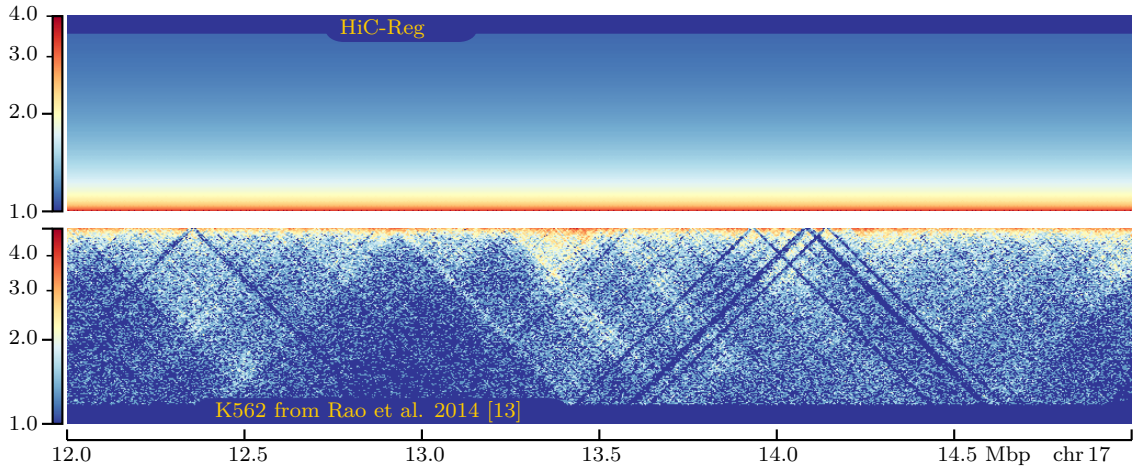
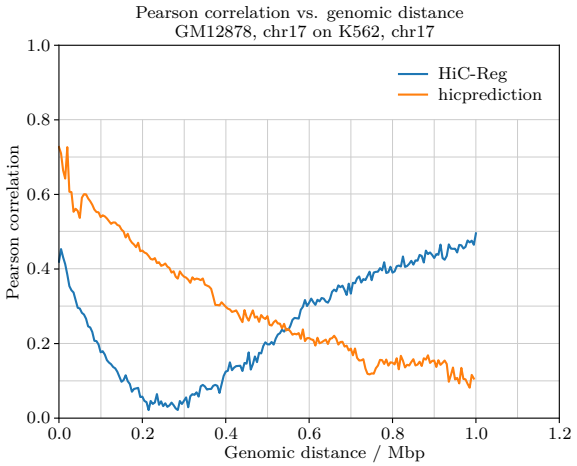
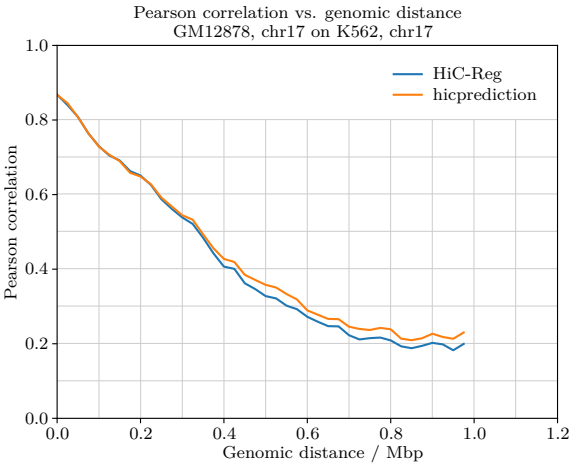


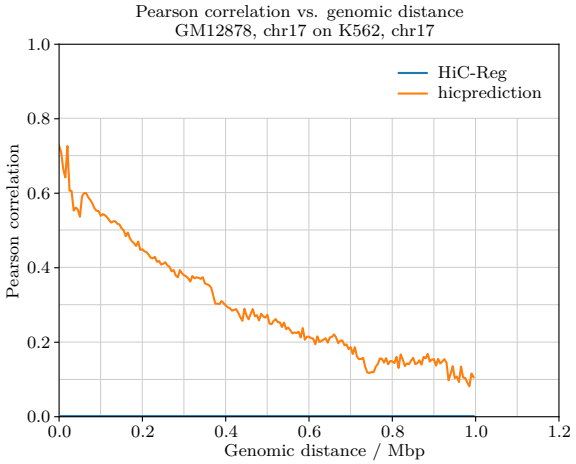
Figure 38: HiC-Reg prediction from BAM files, GM12878 on K562, chromosome 17, 5 kbp



(a) HiC-Reg data from publication [2]



(b) HiC-Reg input data converted from hicprediction datasets



(c) HiC-Reg input data computed from BAM files

Figure 39: Pearson correlation comparison HiC-Reg vs. hicprediction

6 Discussion

Throughout this masterproject, hicprediction as been amended in many ways, and several ideas for improving predictions have been investigated.

On the *input side*, hicprediction now supports protein inputs in broadPeak- and bigwig formats, besides the originally used narrowPeak files, as well as a combination of all three. Additionally, the protein inputs can be smoothed using a Gaussian kernel. With regard to *dataset creation and training*, it is now possible to remove empty samples, remove samples with a distance smaller than a certain threshold and emphasize samples based on interaction count, feature value or TADs. Furthermore, protein signal values and interaction counts can be scaled to arbitrary value ranges, extra trees can be used in addition to random forests, and training datasets from different cell lines can be concatenated. Moreover, cross-validation can now be performed as proposed by Zhang and colleagues [2]. With regard to *predictions*, it is now possible to smoothen the predicted matrix.

However, most of the changes did not improve the predictions significantly with respect to the main metrics, distance stratified Pearson correlation, and in most cases, the predicted matrices also did not improve visually.

Higher-density inputs, like bigwig files, have been shown to reduce "inverted-triangle"- and gradient-style-errors in the predictions, but there were also regions where the predictions from narrowPeak files were actually more informative. To this end, the combination of narrowPeak- and bigwig-inputs might be a useful one, but would need further investigations. Smoothing the protein inputs also worked surprisingly well in some regions, but was not able to fill all empty bins when using sparse input data (narrowPeak files), at least not in the simple setting investigated within this masterproject.

Removing empty samples prevented errors like inverted triangles in the predictions, but also made it impossible to detect interacting regions when sparse inputs like narrowPeak files were used. Conversely, removing empty samples did not change the results much when using higher-density inputs, since most samples were then actually *not* empty.

As expected, Hi-C matrix-scaling did not influence the results much, yet too small values should be avoided to evade problems with "logarithm plus one" representations. Scaling proteins unexpectedly had a negative impact on the resulting predictions. The underlying problem should be investigated further, maybe starting with floating point resolution.

Computing the mean from several single predictions seems to have its merits, as it smoothed out small errors and noise which occurred only in one (or few) of the single predictions. In the example used throughout this masterproject report, the difference between the best single prediction and the mean prediction was small; the averaged prediction actually even looked a bit better. No efforts have been made to check whether this generalizes to other cell lines. However, it is generally recommendable to always compute predictions from all available (training-)cell lines, and then the overhead for computing the mean from the predicted matrices should be acceptable. It would also be conceivable to look into the single predictions first and then compute the mean only from those predictions which actually have some structure. Similar considerations also hold for predictions from concatenated datasets – based on the available results, using them alone

seems questionable, but they may be useful as an addition to the single predictions, again possibly leaving out datasets from cell lines where the single predictions justify doing so. However, training concatenated datasets typically requires considerably more memory than computing the mean of the corresponding cooler matrices, and it is much slower.

The usage of extra trees instead of random forests did not cause major changes in the predictions made for this masterproject; however, this should be verified for other cell lines and chromosomes. If the results proof comparable across all cell lines, then using extra trees might be favorable, because computing *random* split points is usually faster than computing the *best* split points, as required by random forests.

Emphasizing samples with interaction count or protein feature values in a certain range as well as emphasizing samples within TADs did not have the anticipated positive effect on the predictions. It is not yet clear whether these sample-weighting approaches do not work in general or whether their failure was due e. g. to inappropriate parameter search spaces. Also, the objective function has been selected in a rather ad-hoc fashion and might be inappropriate, since cross-validation results were always quite good, but did not – and still do not – generalize very well to cross-cell predictions. Removing samples with distances smaller than 5 kbp can also not be considered successful, but its influence on the results seemed generally stronger than the one from the sample weighting approaches above. While removing all samples within a certain distance might not be useful, dropping outliers might be, especially when adjusting the value range *after* removing outliers.

The comparison between HiC-Reg and hicprediction supports the notion that the prediction results strongly depend on the inputs. Using (converted) inputs from hicprediction in HiC-Reg, the comparatively good results from the publication [2] could not be confirmed. Instead, the predictions were very similar to the ones from hicprediction, the latter however being much faster and requiring considerably less memory. In this regard, it should also not be forgotten that hicprediction is using two proteins fewer than HiC-Reg, because data for RAD21 and TBP was not available in the required input formats, cf. subsection 4.1.

To comprehend, at least three tasks have to remain for future research. First, there are meanwhile a vast number of tuning parameters for hicprediction, so figuring out useful combinations and testing them within a reasonable timeframe has become a challenging task. Within this masterproject, usually only one parameter has been tuned at a time, leaving open the question whether various combinations would yield better results. Secondly, the generalization of learned models to different cell lines seems still insufficient, despite acceptable cross-validation results on the training datasets. Investing time in improving generalization, whether from the input or the models' side, would therefor certainly pay off. Thirdly, selecting and possibly filtering inputs such that they are on the one hand dense enough to allow predicting a reasonable portion of the interaction count matrix, but on the other hand are still indicative of DNA-DNA interactions remains a challenge. Looking at the comparison between the published results from HiC-Reg and the ones from hicprediction in subsection 5.7, this task seems the most important.

Appendices

Figures

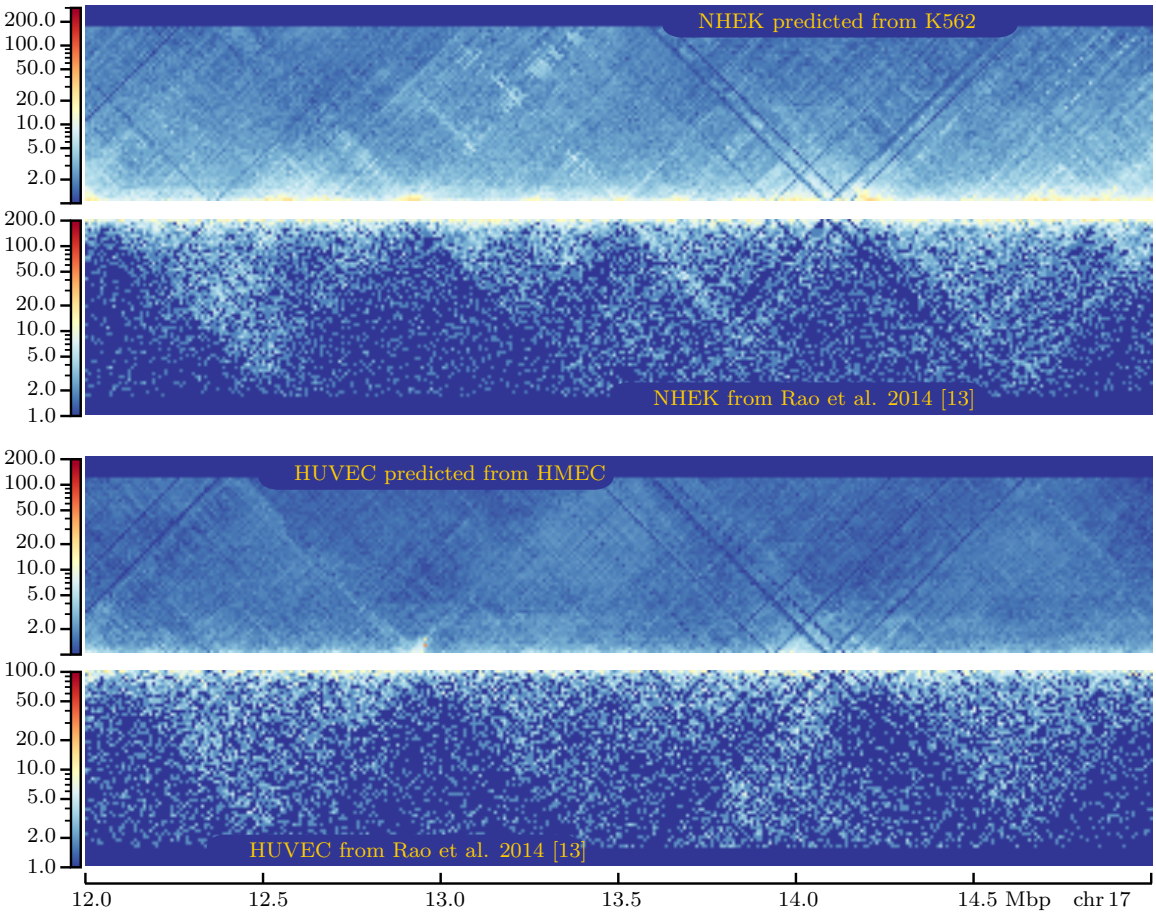


Figure 40: Example predictions from bigwig, all samples

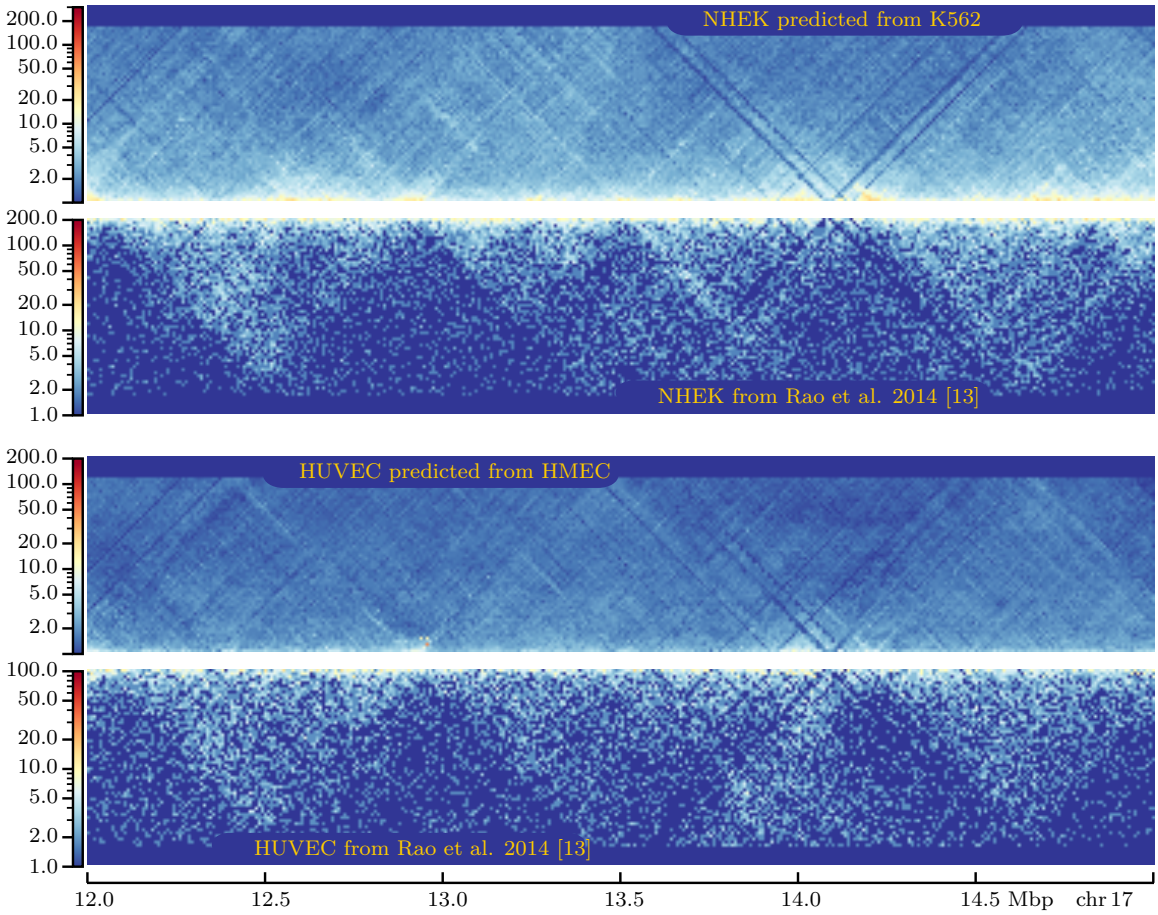
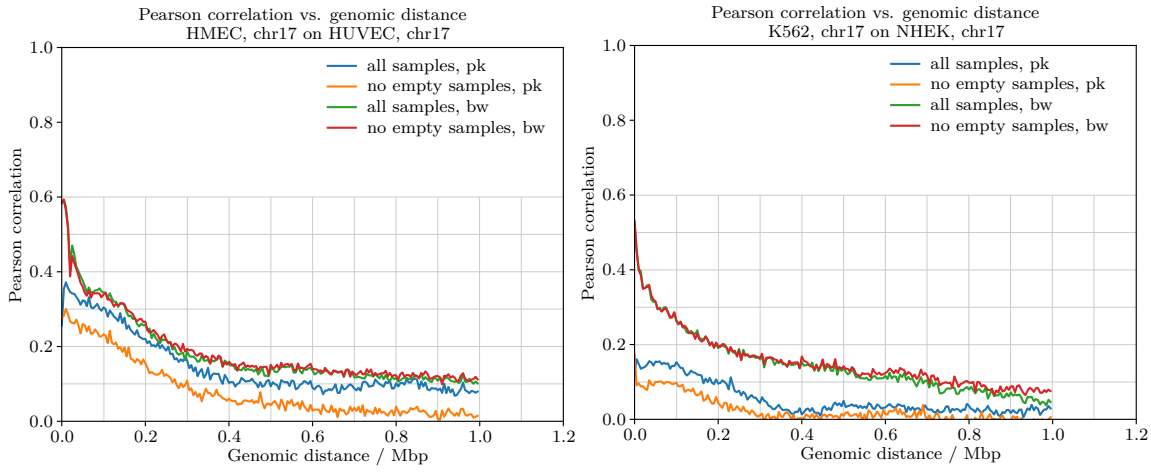


Figure 41: Example predictions from bigwig, no empty samples



(a) HMEC on HUVEC, 5 kb

(b) K562 on NHEK, 5 kb

Figure 42: Pearson correlations for further cell lines

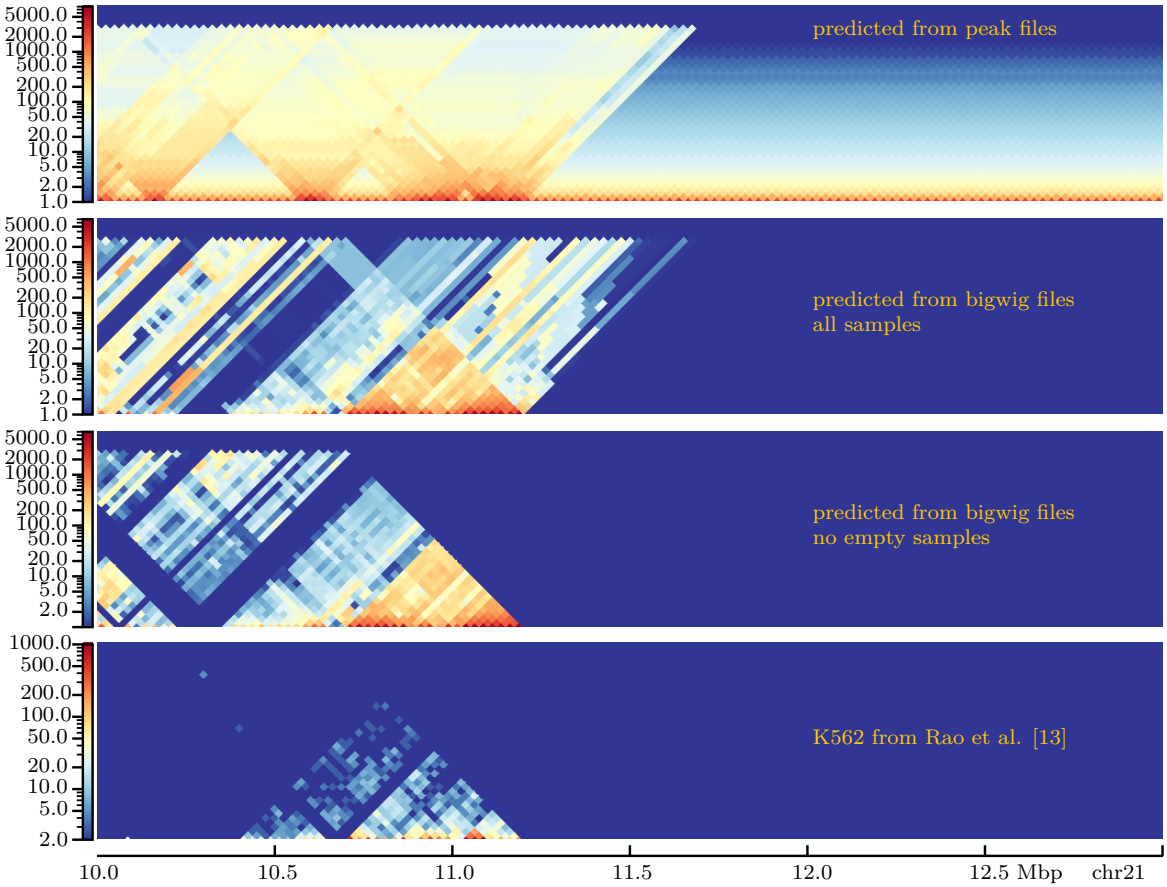


Figure 43: Predictions on chr21 where removing empty samples is better

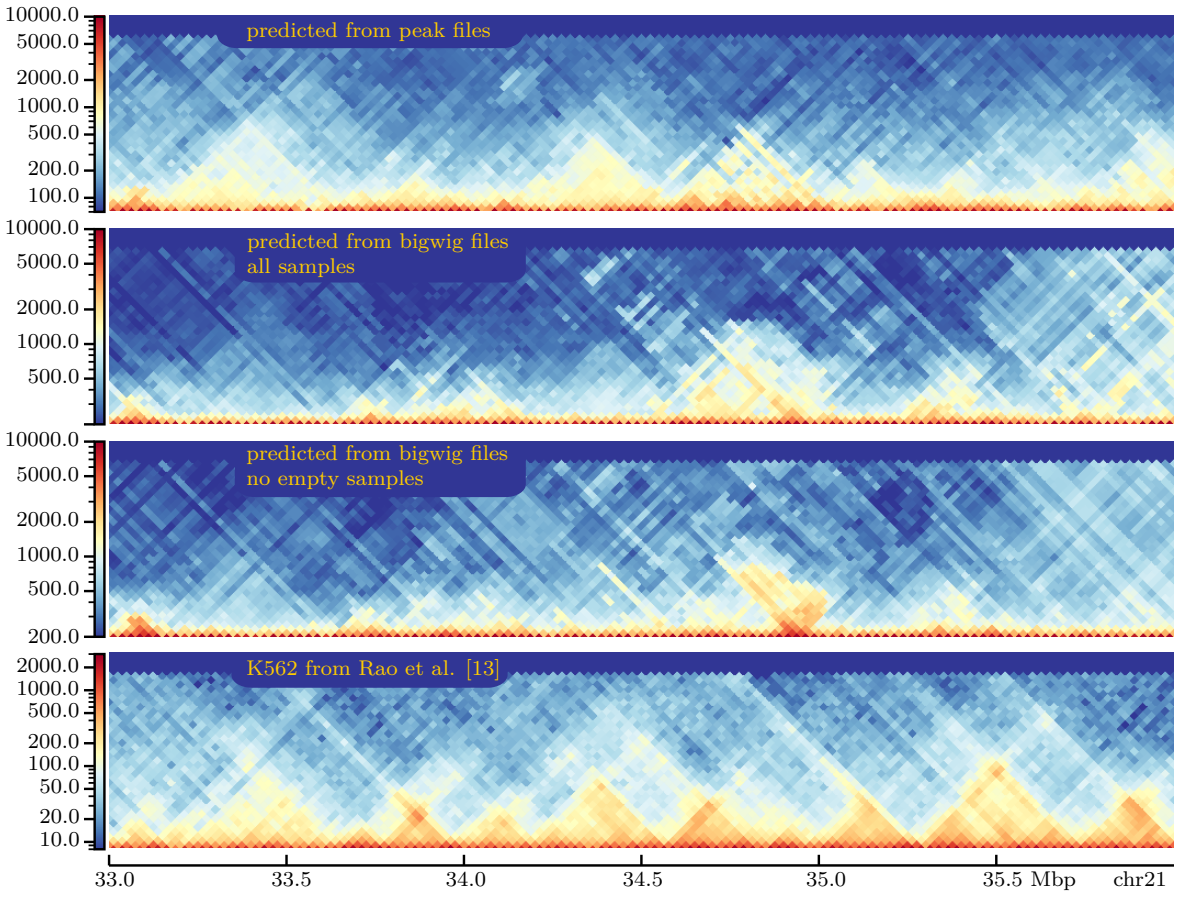


Figure 44: Predictions on chr21 where removing empty samples is worse

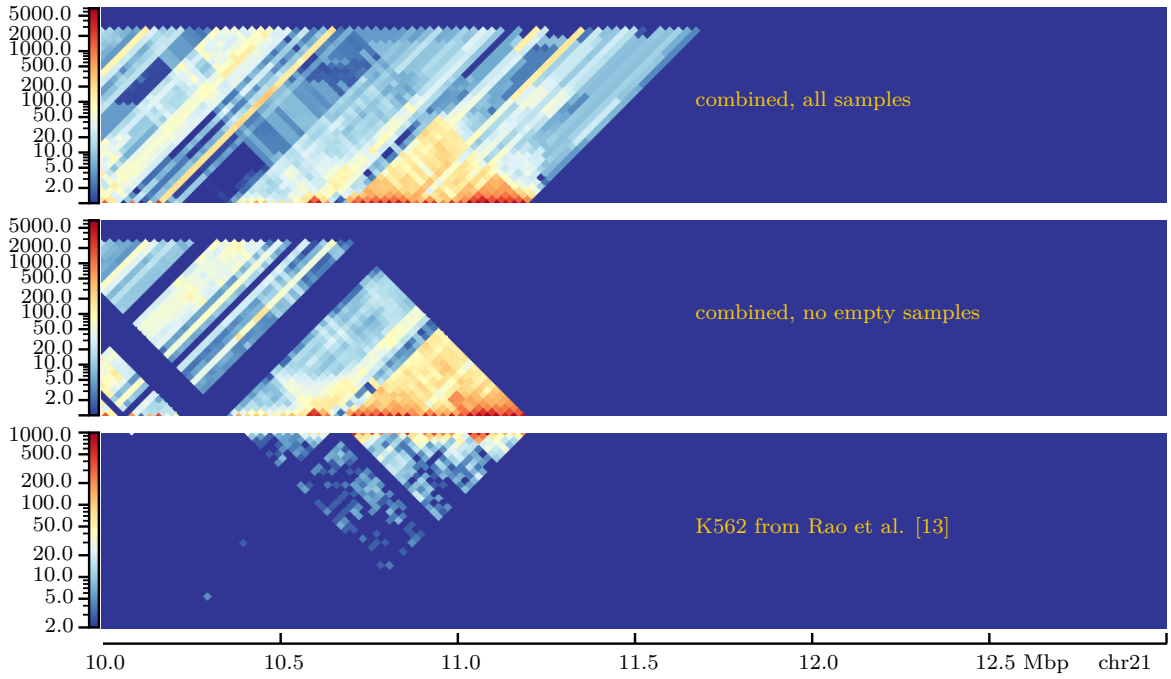


Figure 45: combi prediction chr21 – few interactions

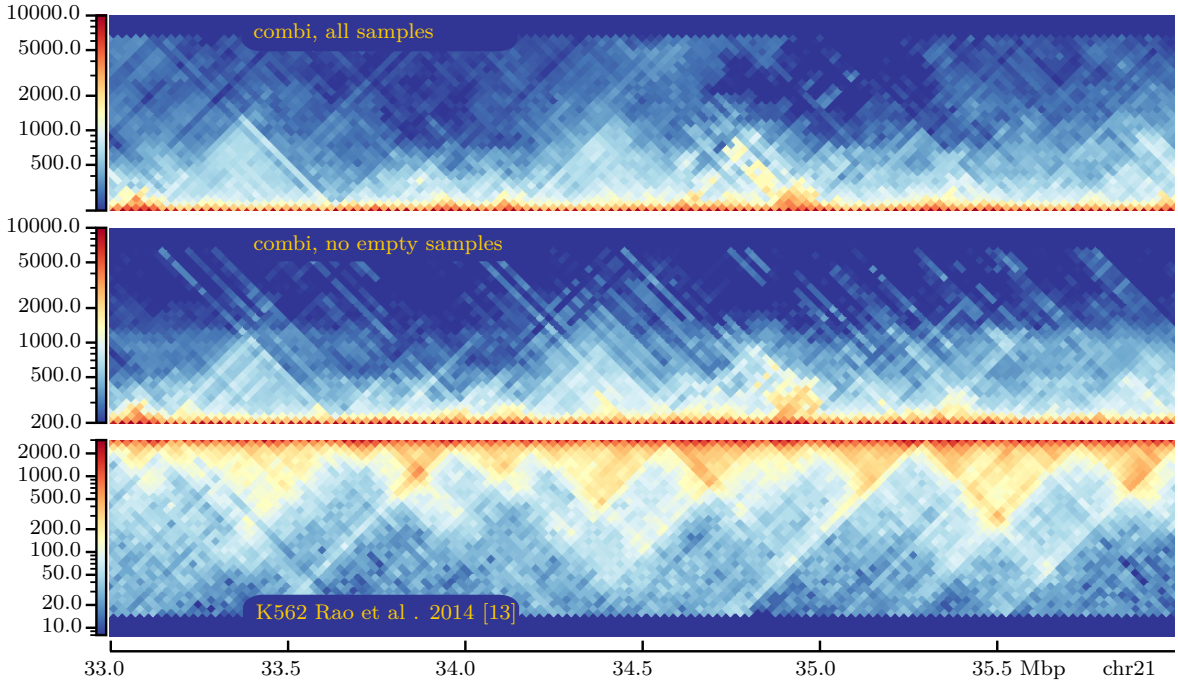


Figure 46: combi prediction chr21 – many interactions

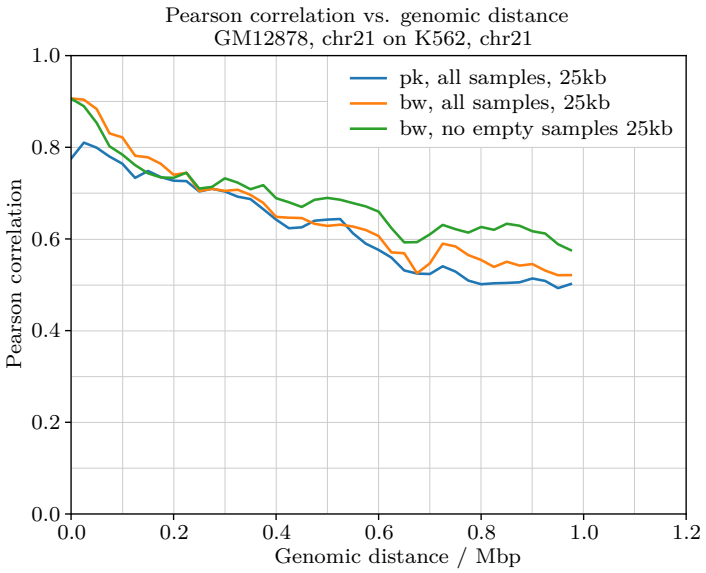
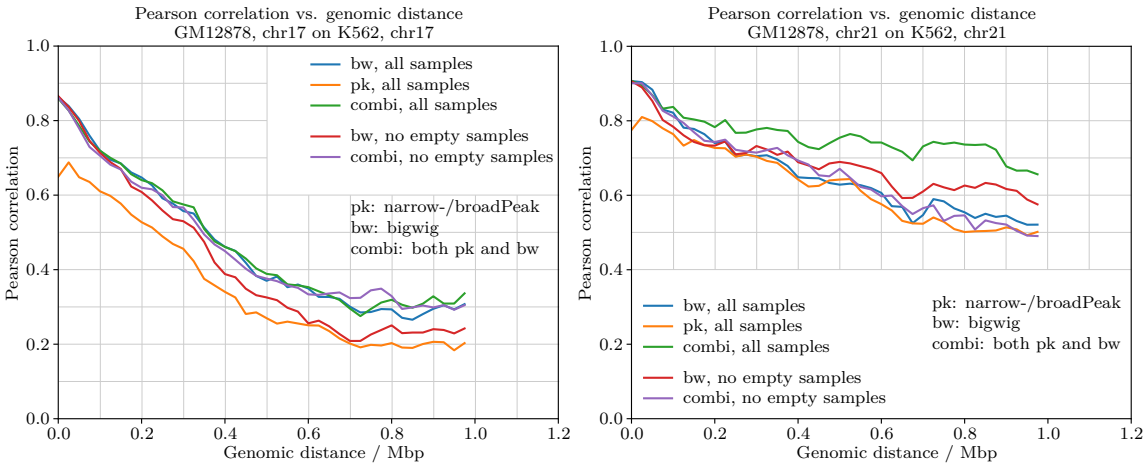


Figure 47: Pearson correlation, 25kb, GM12878 on K562



(a) chromosome 17

(b) chromosome 21

Figure 48: Pearson correlation for combined predictions, GM12878 on K562

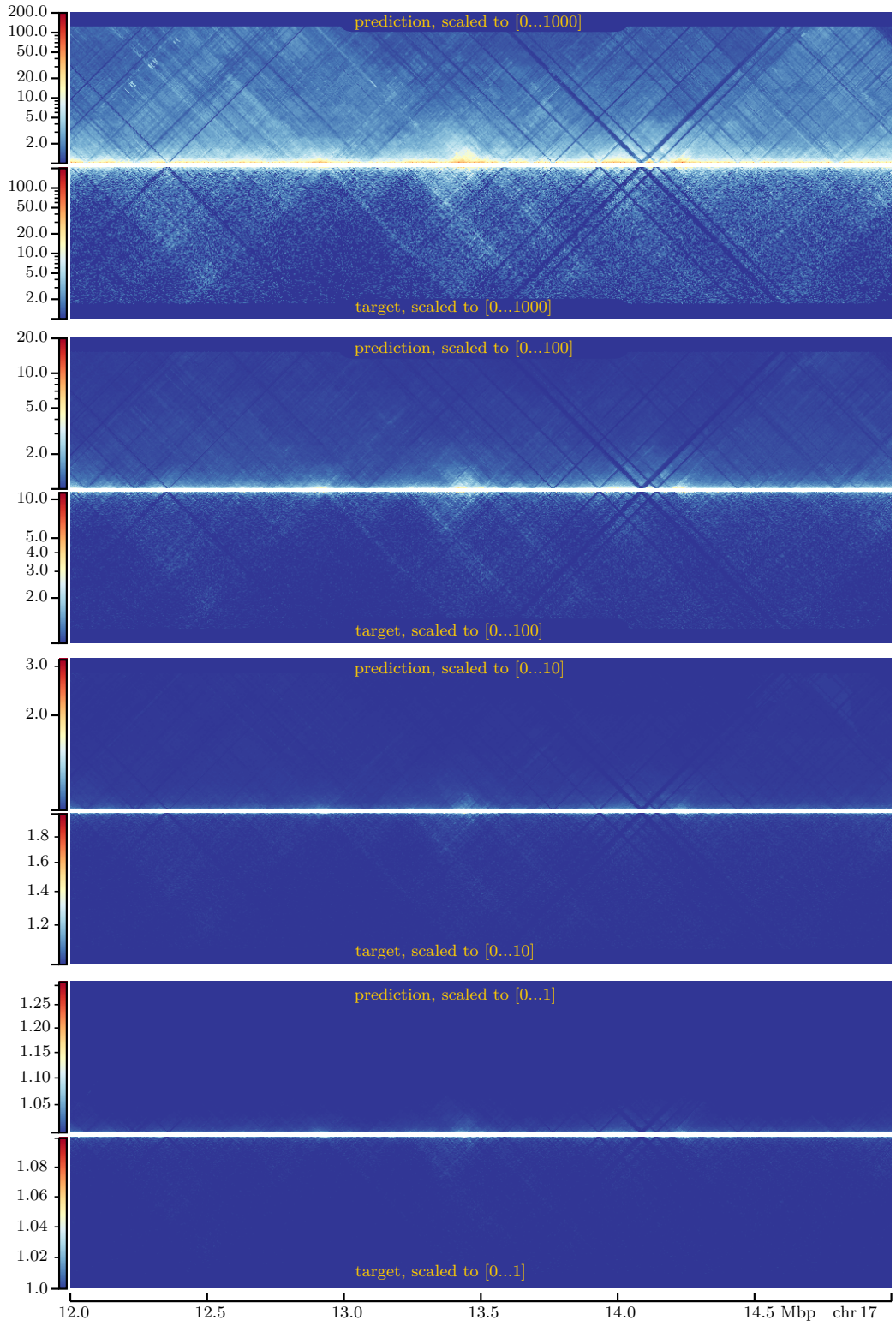


Figure 49: Predictions GM12878 on K562 with scaled matrices

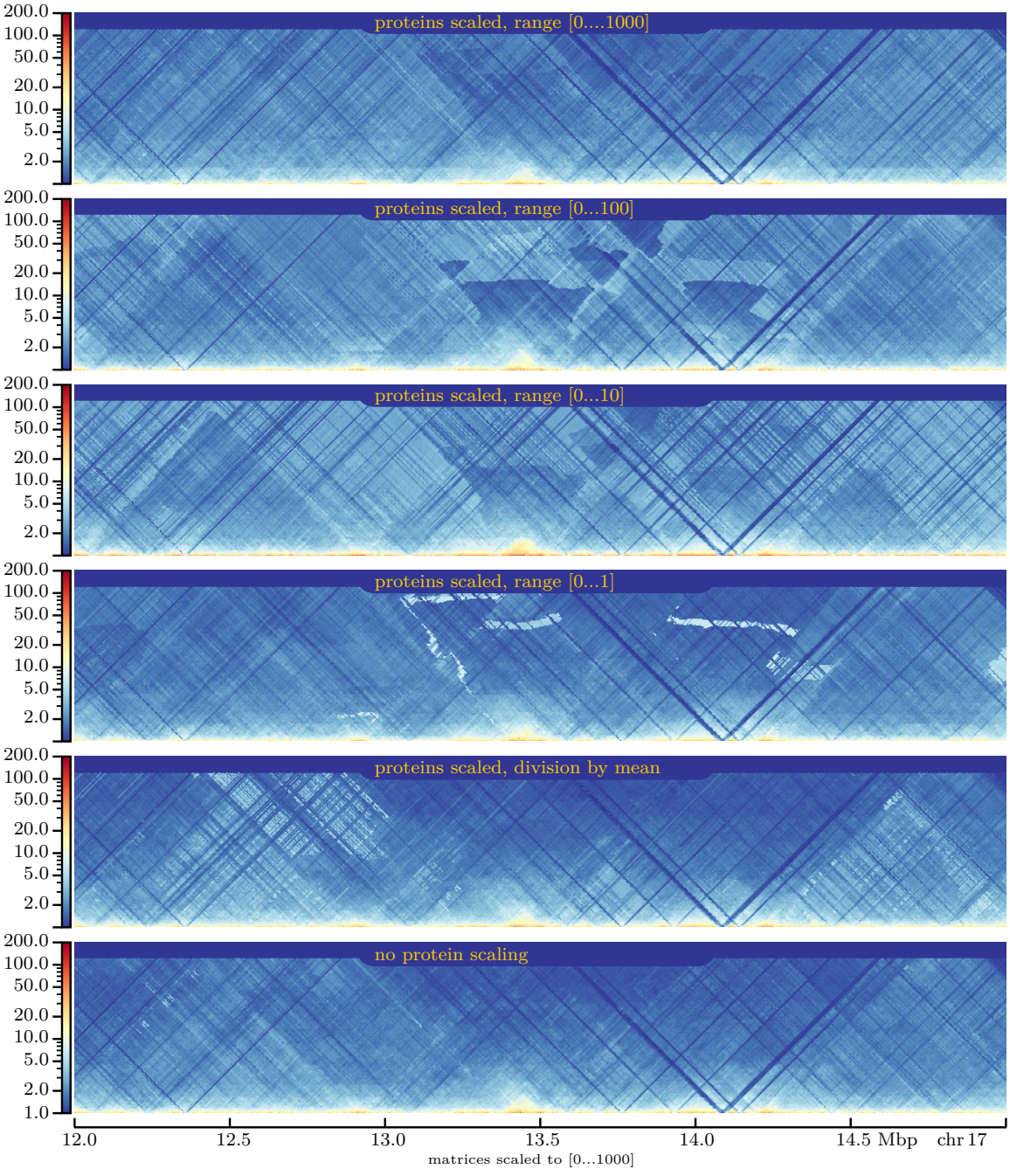


Figure 50: Predictions GM12878 on K562 with scaled proteins

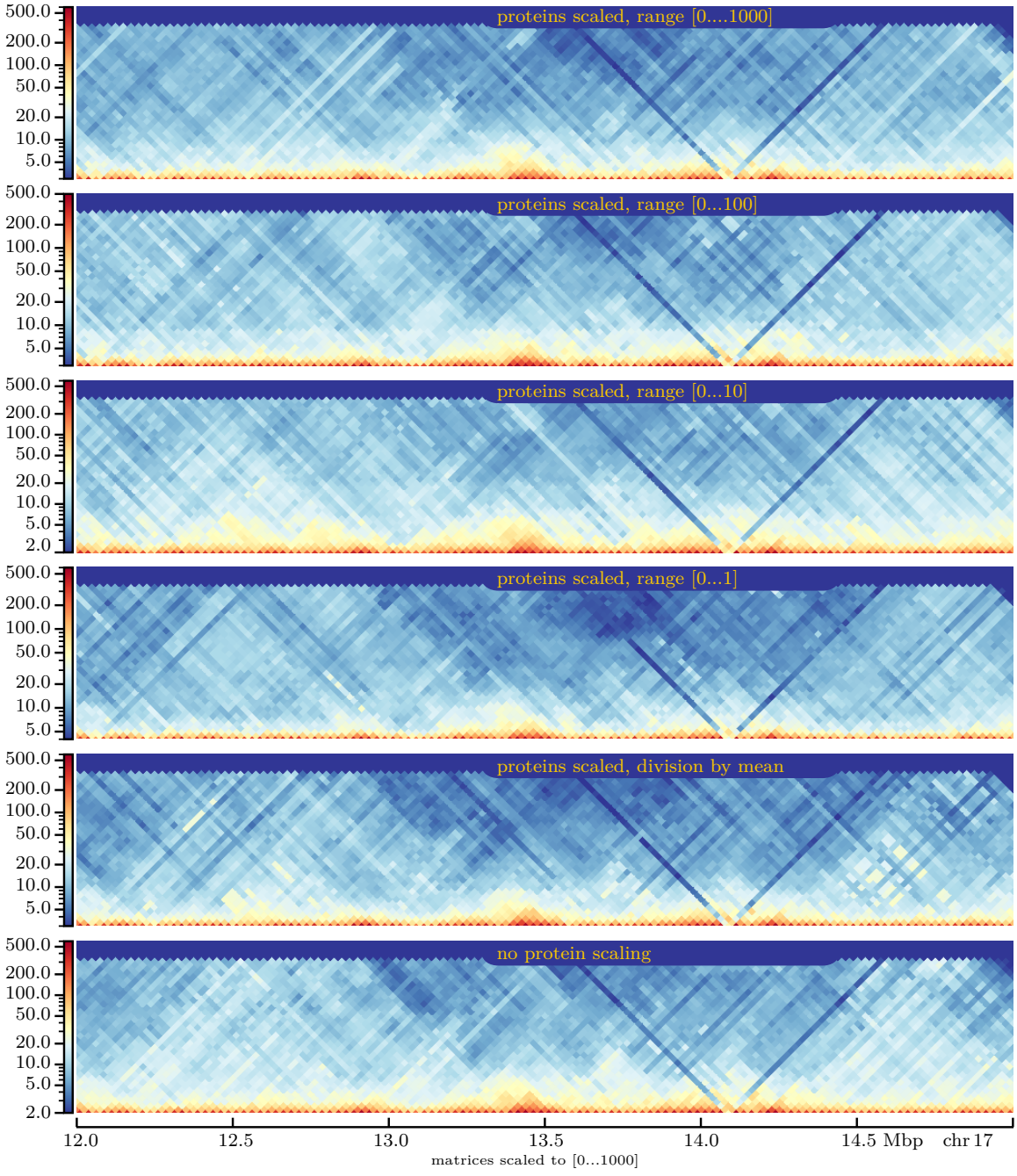
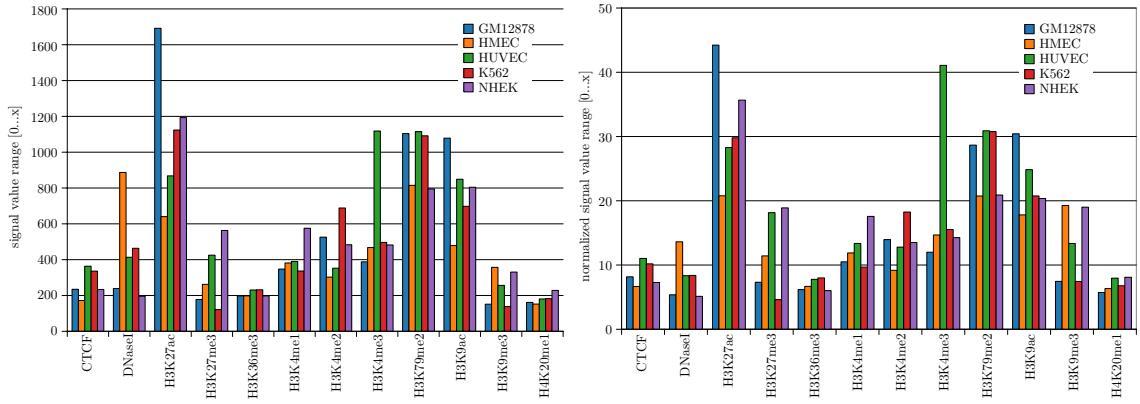


Figure 51: Predictions GM12878 on K562 with scaled proteins



(a) before scaling (b) division-by-mean scaling

Figure 52: Protein signal value ranges for chromosome 17, 25 kbp

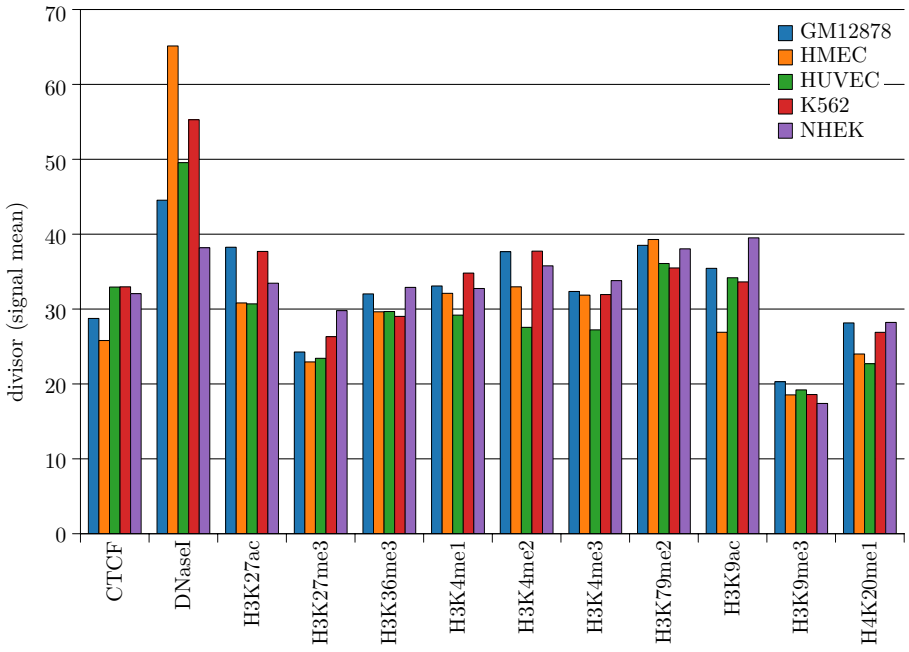


Figure 53: chr17 divison factors / mean signal values, 25 kbp

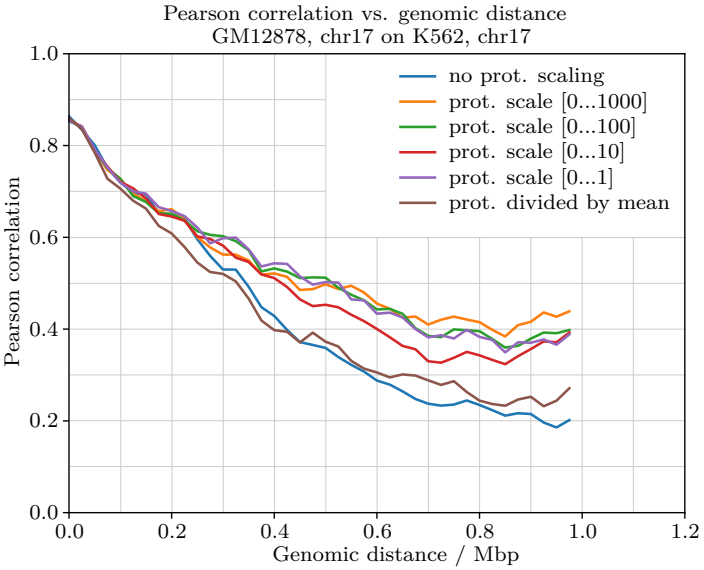


Figure 54: Pearson correlation, 25kb, GM12878 on K562

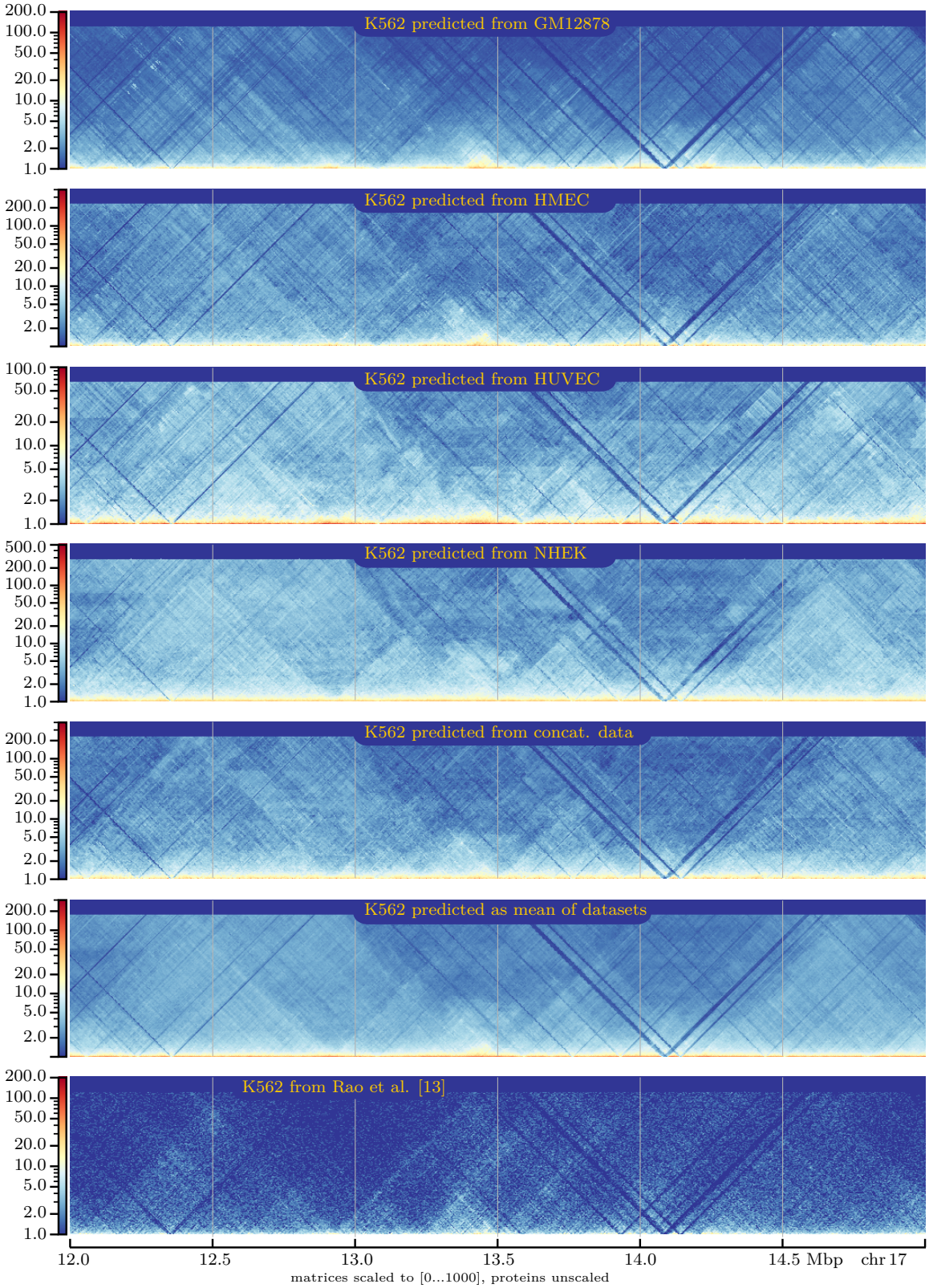


Figure 55: Predictions on K562, single vs. joint datasets

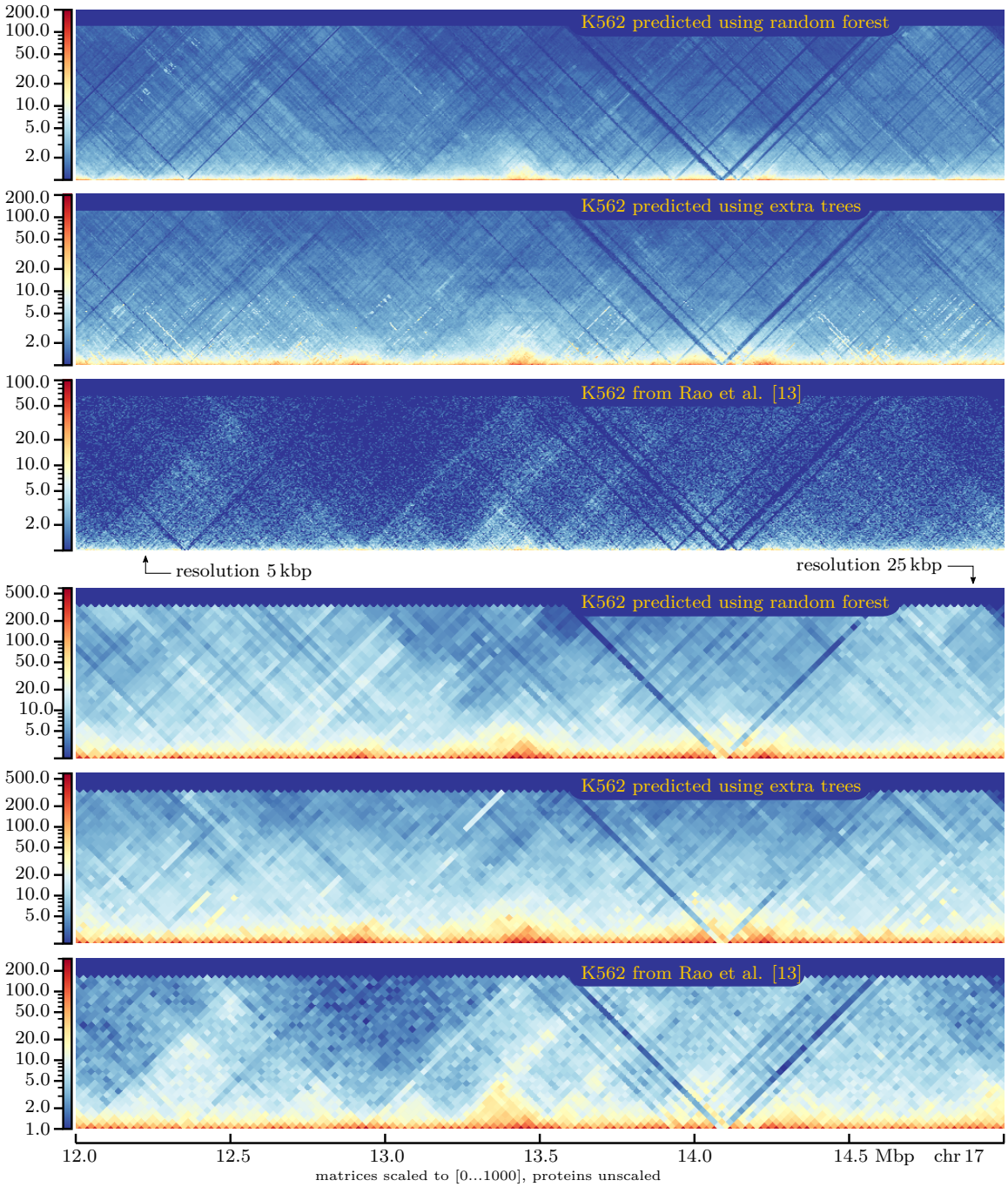


Figure 56: matrix comparison: random forest vs. extra trees

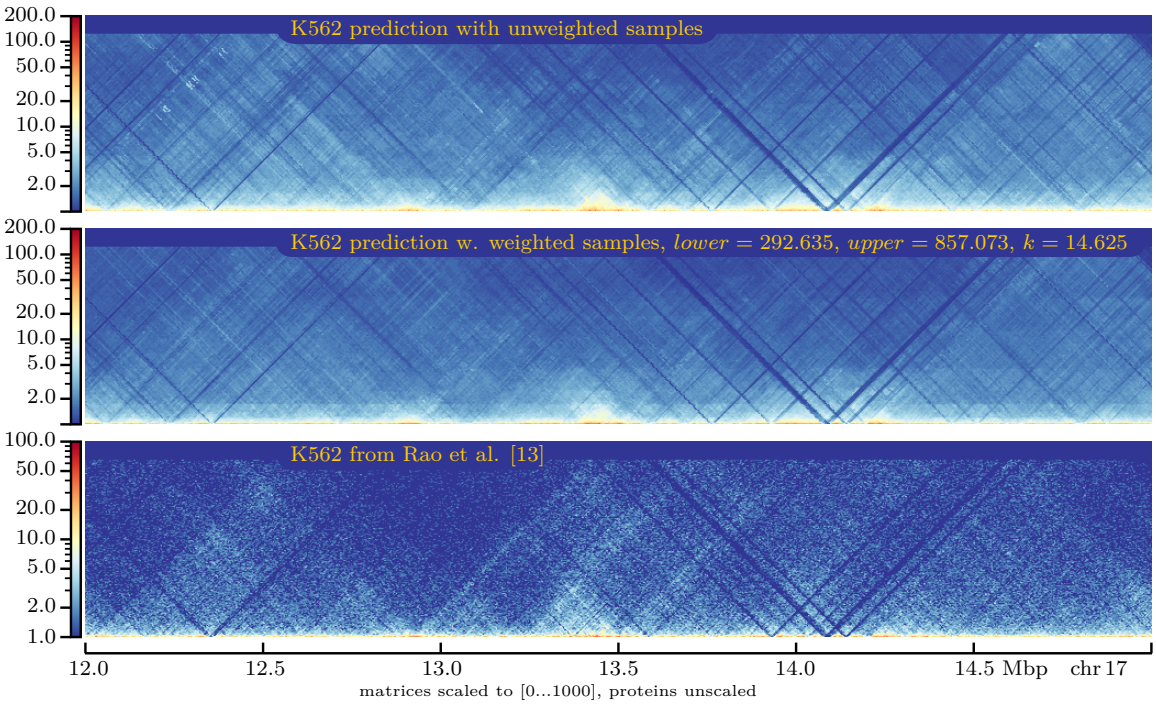


Figure 57: matrix comparison: interaction-count-based sample weighting vs. normal prediction

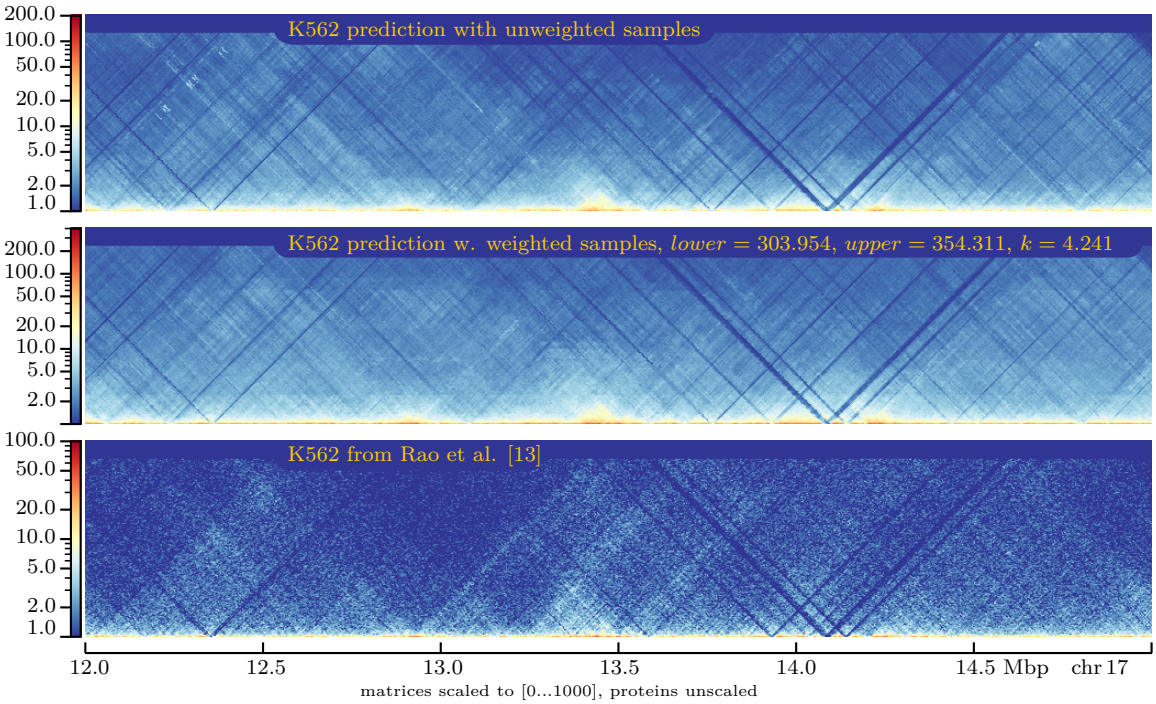


Figure 58: matrix comparison: CTCF-based sample weighting vs. normal prediction

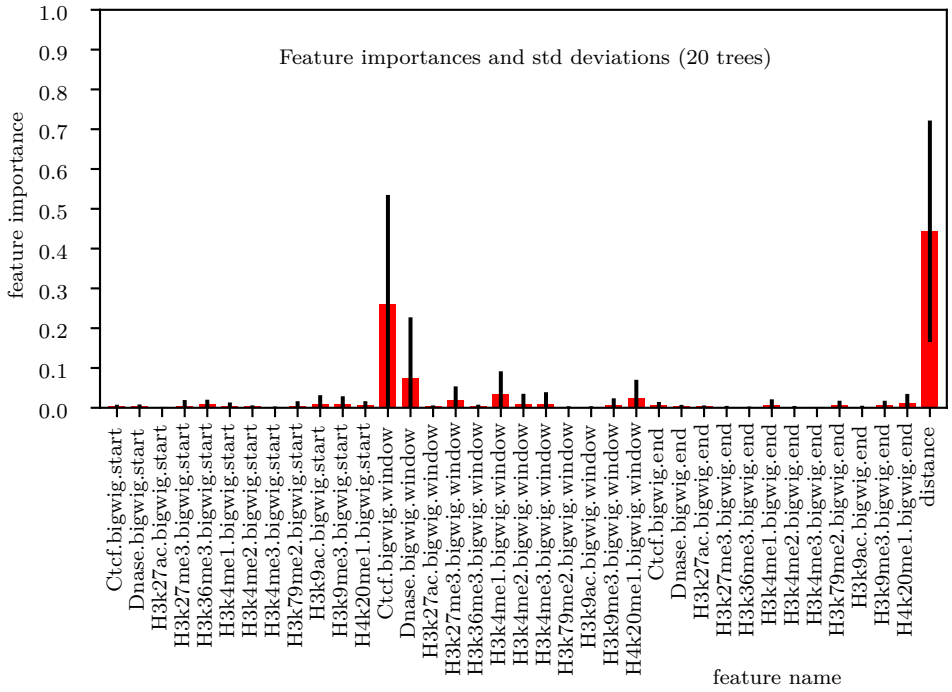


Figure 59: feature importances CTCF-based sample weighting

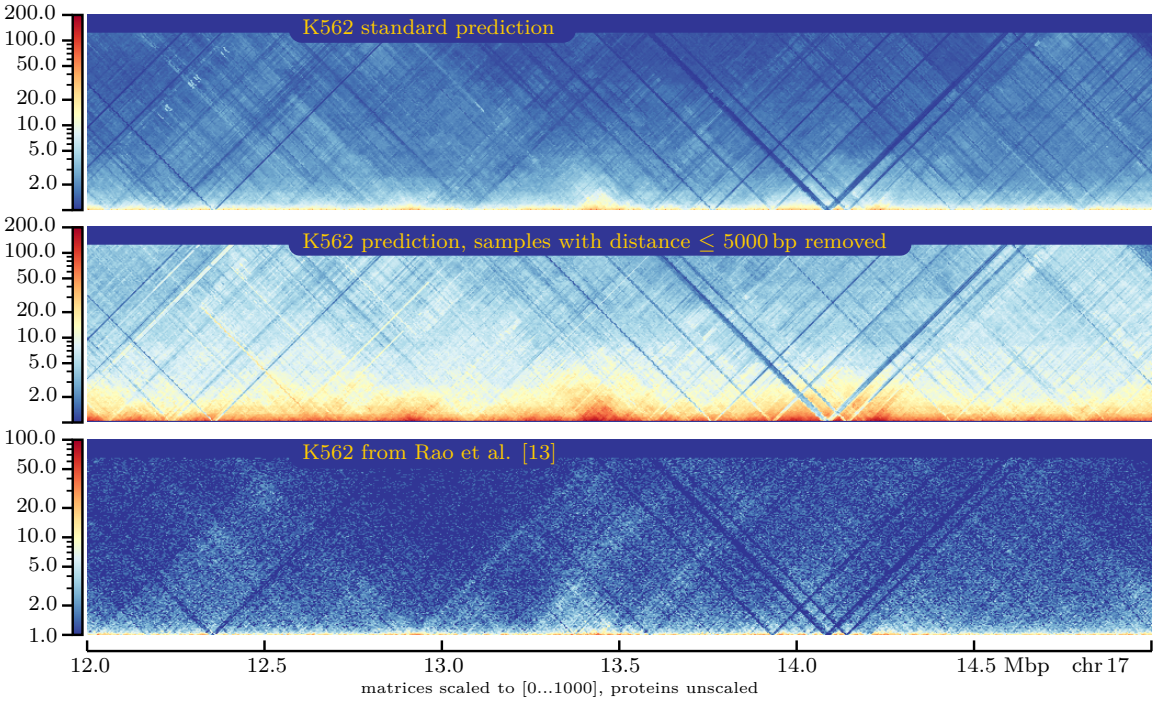


Figure 60: matrix comparison: samples ≤5 kbp removed vs. normal prediction

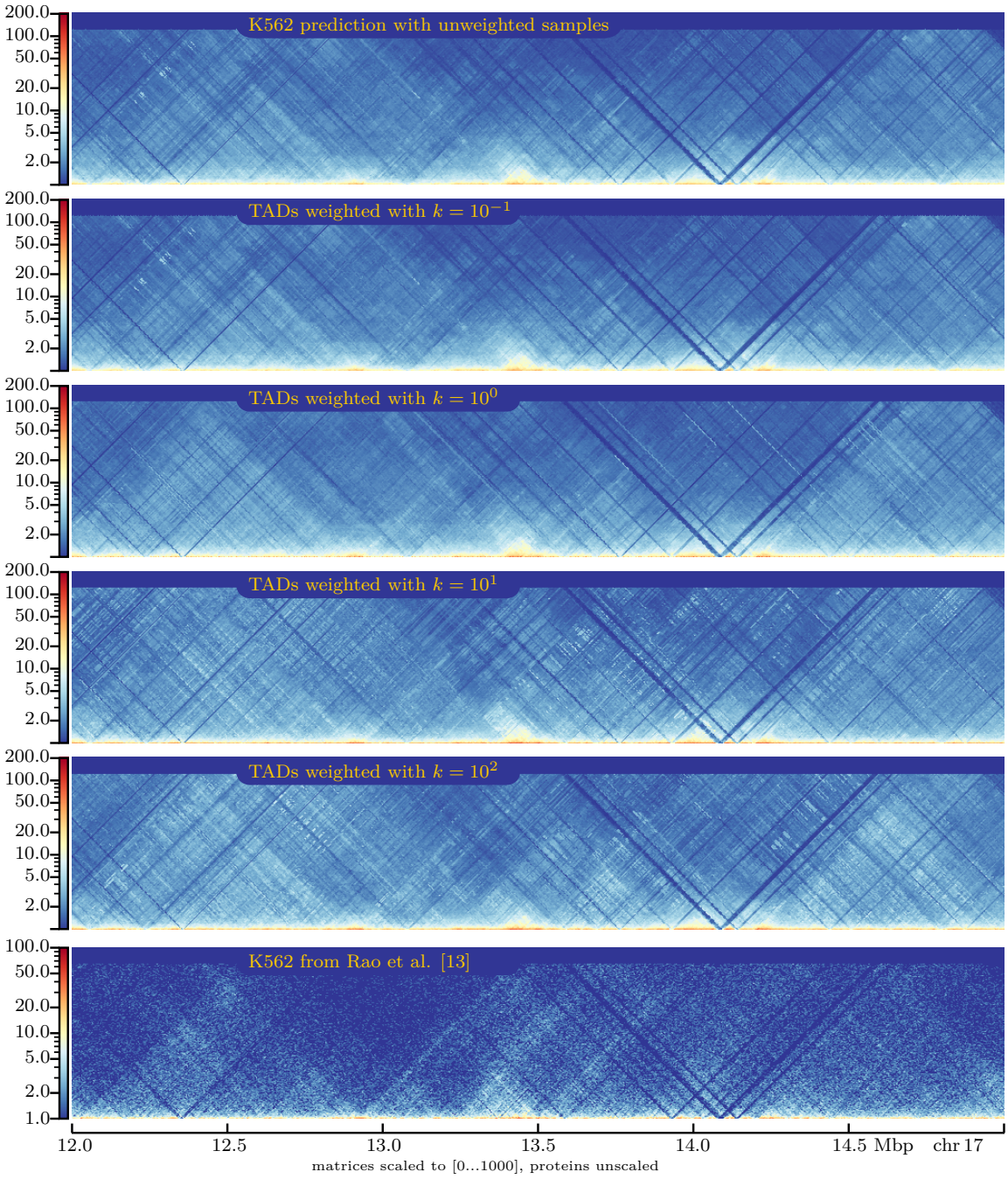


Figure 61: matrix comparison: TAD-based sample weighting vs. normal prediction

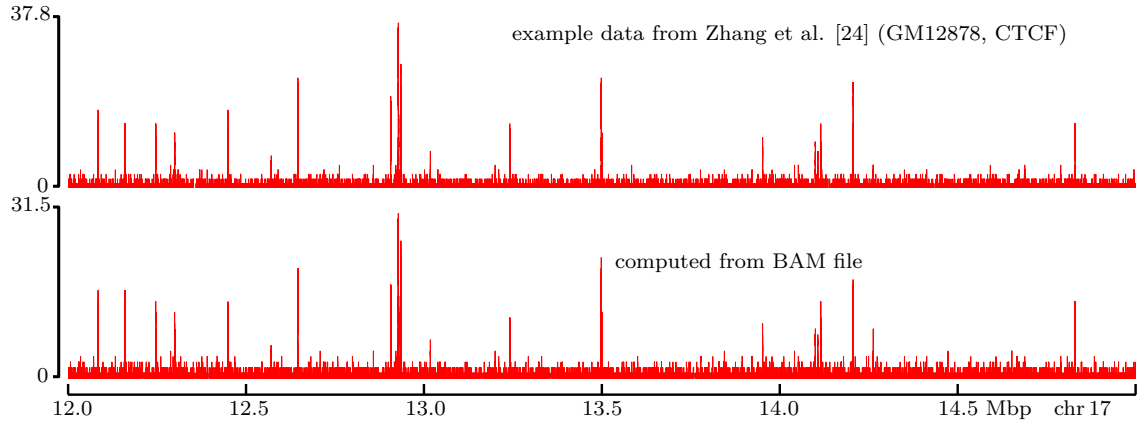


Figure 62: converted input data vs. example data from HiC-Reg github, 3 Mbp

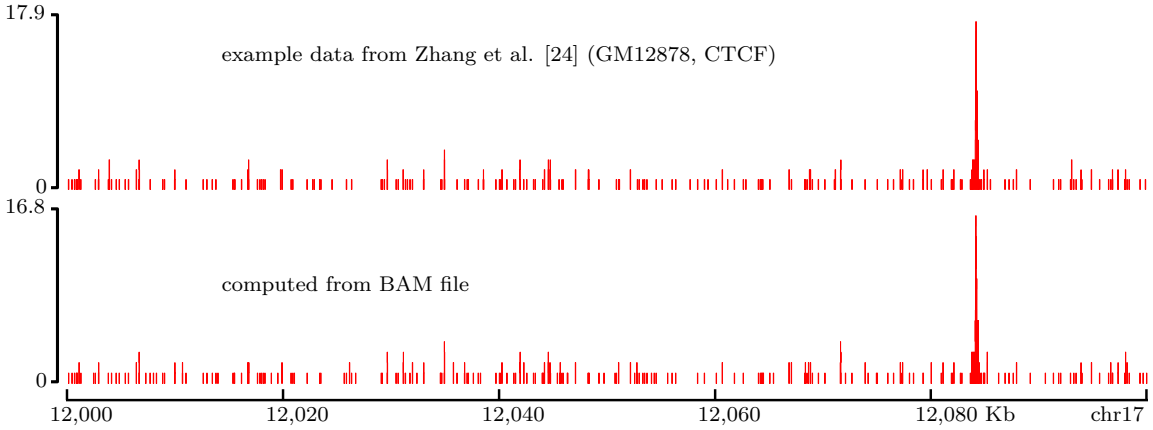


Figure 63: converted input data vs. example data from HiC-Reg github, 100 kbp

Listings

```
#bash-style code
#indexing a bam file
samtools index ${BAMFILE} ${BAMFILE%bam}.bai
#creating a bigwig file from the bam file above
OUTFILE="${BAMFILE%bam}bigwig"
hg19SIZE="2685511504"
COMMAND="--numberOfProcessors 10 --bam ${BAMFILE}"
COMMAND="${COMMAND} --outFileName ${OUTFILE}"
COMMAND="${COMMAND} --outFileFormat bigwig"
COMMAND="${COMMAND} --binSize 5000 --normalizeUsing RPGC"
COMMAND="${COMMAND} --effectiveGenomeSize ${hg19SIZE}"
COMMAND="${COMMAND} --scaleFactor 1.0 --extendReads 200"
COMMAND="${COMMAND} --minMappingQuality 30"
bamCoverage ${COMMAND}
#computing mean from replicate 1 and 2 bigwig files
REPLICATE1="${FOLDER1}${PROTEIN}.bigwig"
REPLICATE2="${FOLDER2}${PROTEIN}.bigwig"
OUTFILE="${OUTFOLDER}${PROTEIN}.bigwig"
COMMAND="-b1 ${REPLICATE1} -b2 ${REPLICATE2}"
COMMAND="${COMMAND} -o ${OUTFILE} -of bigwig"
COMMAND="${COMMAND} --operation mean -bs 5000"
COMMAND="${COMMAND} -p 10 -v"
bigwigCompare ${COMMAND}
```

Listing 1: bam to bigwig

```
  #default parameters for the random forest
  #(trying to match HiC-Reg)
  modelParamDict = dict(n_estimators=20,
                        criterion='mse',
                        max_depth=None,
                        min_samples_split=2,
                        min_samples_leaf=1,
                        min_weight_fraction_leaf=0.0,
                        max_features=1./3.,
                        max_leaf_nodes=None,
                        min_impurity_decrease=0.0,
                        min_impurity_split=None,
                        bootstrap=True,
                        oob_score=False,
                        n_jobs=-1,
                        random_state=5,
                        verbose=2,
                        ccp_alpha=0.0,
                        max_samples=0.75)
  # ... ... ...
  #train(..., pModelParamDict= modelParamDict, ...)
  # ... ... ...
  model = sklearn.ensemble.RandomForestRegressor(**pModelParamDict)
```

Listing 2: hicprediction random forest

```
#sum up the four single predictions for cell line CL
#pred-On-CL_1 to _4 stand for the predicted
#cooler matrices from the four training cell lines
```

```
cmd="--matrices"
cmd="${cmd} ${pred-On-CL_1} ${pred-On-Cl_2}"
cmd="${cmd} ${pred-on-CL_3} ${pred-On-CL_4}"
cmd="${cmd} -o summed-all-on-CL.cool"
hicSumMatrices ${cmd}
#divide the summed matrix by 4
cmd="-m summed-all-on-CL.cool"
cmd="${cmd} --normalize multiplicative"
cmd="${cmd} --multiplicativeValue 0.25"
cmd="${cmd} -o mean-all-on-CL.cool"
hicNormalize ${cmd}
```

Listing 3: compute mean prediction

```
cmd="-m GSE63525_GM12878_combined_30_5kb.cool"
cmd="${cmd} --minDepth 80000"
cmd="${cmd} --maxDepth 160000"
cmd="${cmd} --correctForMultipleTesting fdr"
cmd="${cmd} --outPrefix ${destinationFolder}"
hicFindTADs ${cmd}
```

Listing 4: command for finding TADs on GM12878

```
# prepare interaction count file e.g. for CTCF
chromosome="chr17"
protName="CTCF"
tmpfile="${protName}_${chromosome}.bam"
countfile="${tmpfile%bam}count"
outfile="${tmpfile%bam}txt"
cmd="${protName}.bam ${chromosome}"
samtools view -b ${cmd} > ${tmpfile}
cmd="-ibam ${tmpfile} -bg"
bedtools genomecov ${cmd} > ${countfile}
# $countfile is in bedgraph format

# bin the protein signals
# no specification could be found for the region file
# so just used the example
regionfile="Scripts/aggregateSignalInRegion/hg19_5kbp_chr17.txt"
chromsizefile="hg19.fa.fai"
cmd="${regionfile} ${chromsizefile} ${countfile} ${outfile}"
./aggregateSignal ${cmd}
# outfile is in tab-separated text format
```

Listing 5: preparing inputs for HiC-Reg (example)

```
# text file which holds file names for all 12 protein features
featurefiles="textfileWithNamesOfAllProteinInputs.txt"
# tab-separated Hi-C matrix created by convertForHicReg.py
matrixfile="GM12878matrixConvertedFromHicprediction.txt"
outfolder="out/"
cmd="${matrixfile}"
cmd="${cmd} 1000000 5 regionwise ${featurefiles}"
cmd="${cmd} no ${outfolder} yes Window"
./genDatasetsRH ${cmd}
```

Listing 6: computing window features for HiC-Reg (example)

```
/usr/bin/time -p -v -o $OUTFILE $COMMAND $ARGS
```

Listing 7: measuring ressource consumption

References

[1] Shilu Zhang et al. “In silico prediction of high-resolution Hi-C interaction matrices”. In: *bioRxiv* (Sept. 2018). DOI: 10.1101/406322.

[2] Shilu Zhang et al. “In silico prediction of high-resolution Hi-C interaction matrices”. In: *Nature Communications* 10.1 (Dec. 2019), pp. 1–18. DOI: 10.1038/s41467-019-13423-8.

[3] Andre Bajorat. *Hi-C Predictions based on protein levels*. Tech. rep. University of Freiburg, Sept. 23, 2019. URL: https://www.bioinf.uni-freiburg.de/Lehre/Theses/TP_Andre_Bajorat.pdf.

[4] J. D. Watson and F. H. C. Crick. “Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid”. In: *Nature* 171.4356 (Apr. 1953), pp. 737–738. DOI: 10.1038/171737a0.

[5] Francis H. C. Crick. “On protein synthesis”. In: *Symposia of the Society for Experimental Biology*. Vol. 12. 138-63. 1958, p. 8.

[6] Ana Pombo and Niall Dillon. “Three-dimensional genome architecture: players and mechanisms”. In: *Nature reviews Molecular cell biology* 16.4 (2015), p. 245. DOI: 10.1038/nrm3965.

[7] Boyan Bonev and Giacomo Cavalli. “Organization and function of the 3D genome”. In: *Nature Reviews Genetics* 17.11 (2016), p. 661. DOI: 10.1038/nrg.2016.112.

[8] J. Dekker. “Capturing Chromosome Conformation”. In: *Science* 295.5558 (Feb. 2002), pp. 1306–1311. DOI: 10.1126/science.1067799.

[9] Marieke Simonis et al. “Nuclear organization of active and inactive chromatin domains uncovered by chromosome conformation capture–on-chip (4C)”. In: *Nature Genetics* 38.11 (Oct. 2006), pp. 1348–1354. DOI: 10.1038/ng1896.

[10] J. Dostie et al. “Chromosome Conformation Capture Carbon Copy (5C): A massively parallel solution for mapping interactions between genomic elements”. In: *Genome Research* 16.10 (Oct. 2006), pp. 1299–1309. DOI: 10.1101/gr.5571506.

[11] Erez Lieberman-Aiden et al. “Comprehensive mapping of long-range interactions reveals folding principles of the human genome”. In: *Science* 326 (2009), pp. 289–293. DOI: 10.1126/science.1181369.

[12] Nynke L. van Berkum et al. “Hi-C: A Method to Study the Three-dimensional Architecture of Genomes.” In: *Journal of Visualized Experiments* 39 (May 2010). DOI: 10.3791/1869.

[13] Suhas S. P. Rao et al. “A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping”. In: *Cell* 159.7 (2014), pp. 1665–1680. DOI: https://doi.org/10.1016/j.cell.2014.11.021.

[14] D. S. Johnson et al. “Genome-Wide Mapping of in Vivo Protein-DNA Interactions”. In: *Science* 316.5830 (June 2007), pp. 1497–1502. DOI: 10.1126/science.1141319.

[15] Gordon Robertson et al. “Genome-wide profiles of STAT1 DNA association using chromatin immunoprecipitation and massively parallel sequencing”. In: *Nature Methods* 4.8 (June 2007), pp. 651–657. DOI: 10.1038/nmeth1068.

[16] Jim R. Hughes et al. “Analysis of hundreds of cis-regulatory landscapes at high resolution in a single, high-throughput experiment”. In: *Nature Genetics* 46.2 (Jan. 2014), pp. 205–212. DOI: 10.1038/ng.2871.

[17] Michele Di Pierro et al. “De novo prediction of human chromosome structures: Epigenetic marking patterns encode genome architecture”. In: *Proceedings of the National Academy of Sciences* 114.46 (Oct. 2017), pp. 12126–12131. DOI: 10.1073/pnas.1714980114.

[18] Pau Farré et al. “Dense neural networks for predicting chromatin conformation”. In: *BMC Bioinformatics* 19.372 (Oct. 2018). DOI: 10.1186/s12859-018-2286-z.

[19] Ron Schwessinger et al. “DeepC: Predicting chromatin interactions using megabase scaled deep neural networks and transfer learning”. In: *bioRxiv* (Aug. 2019). DOI: 10.1101/724005.

[20] Wenran Li, Wing Hung Wong and Rui Jiang. “DeepTACT: predicting 3D chromatin contacts via bootstrapping deep learning”. In: *Nucleic Acids Research* 47.10 (Mar. 2019), e60. DOI: 10.1093/nar/gkz167.

[21] University of California Santa Cruz. *Genome Browser FAQ – narrowPeak*. 2020. URL: https://genome.ucsc.edu/FAQ/FAQformat.html#format12.

[22] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[23] scikit-learn developers. *scikit-learn documentation, section 1.11.: Ensemble Methods*. 2019. URL: https://scikit-learn.org/stable/modules/ensemble.html (visited on 05/05/2020).

[24] S. Roy, S. Zhang and D. Chasman. *HiC-Reg: In silico prediction of high-resolution Hi-C interaction matrices*. 2020. URL: https://github.com/Roy-lab/HiC-Reg (visited on 03/20/2020).

[25] Tao Yang et al. “HiCRep: assessing the reproducibility of Hi-C data using a stratum-adjusted correlation coefficient”. In: *Genome Research* 27.11 (Aug. 2017), pp. 1939–1949. DOI: 10.1101/gr.220640.117.

[26] Nezar Abdennur and Leonid A. Mirny. “Cooler: scalable storage for Hi-C data and other genomically labeled arrays”. In: *Bioinformatics* (July 2019). Ed. by Jonathan Wren. DOI: 10.1093/bioinformatics/btz540.

[27] Ralf Krauth and Andre Bajorat. *hicprediction - createBaseFile.py*. May 31, 2020. URL: https://github.com/MasterprojectRK/HiCPrediction/blob/master/hicprediction/createBaseFile.py.

[28] Ralf Krauth and Andre Bajorat. *hicprediction - createTrainingSet.py*. May 31, 2020. URL: https://github.com/MasterprojectRK/HiCPrediction/blob/master/hicprediction/createTrainingSet.py.

[29] Ralf Krauth and Andre Bajorat. *hicprediction - training.py*. May 31, 2020. URL: https://github.com/MasterprojectRK/HiCPrediction/blob/master/hicprediction/training.py.

[30] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[31] Ralf Krauth and Andre Bajorat. *hicprediction - predict.py*. May 31, 2020. URL: https://github.com/MasterprojectRK/HiCPrediction/blob/master/hicprediction/predict.py.

[32] Andre Bajorat. *hicprediction*. URL: https://github.com/abajorat/HiCPrediction/ (visited on 03/27/2020).

[33] Anaconda, Inc. *Miniconda*. URL: https://docs.conda.io/en/latest/miniconda.html (visited on 03/29/2020).

[34] Carolin Strobl et al. "Bias in random forest variable importance measures: Illustrations, sources and a solution". In: *BMC Bioinformatics* 8.1 (Jan. 2007). DOI: 10.1186/1471-2105-8-25.

[35] Bum-Kyu Lee and Vishwanath R. Iyer. "Genome-wide Studies of CCCTC-binding Factor (CTCF) and Cohesin Provide Insight into Chromatin Structure and Regulation". In: *Journal of Biological Chemistry* 287.37 (Sept. 2012), pp. 30906–30913. DOI: 10.1074/jbc.r111.324962.

[36] Pierre Geurts, Damien Ernst and Louis Wehenkel. "Extremely randomized trees". In: *Machine Learning* 63.1 (Mar. 2006), pp. 3–42. DOI: 10.1007/s10994-006-6226-1.

[37] S. S. P. Rao et al. *GM12878 HiC-matrix*. URL: ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE63nnn/GSE63525/suppl/GSE63525%5FGM12878%5Finsitu%5Fprimary%2Breplicate%5Fcombined%5F30%2Ehic.

[38] S. S. P. Rao et al. *HMEC HiC-matrix*. URL: ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE63nnn/GSE63525/suppl/GSE63525_HMEC_combined_30.hic.

[39] S. S. P. Rao et al. *HUVEC HiC-matrix*. URL: ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE63nnn/GSE63525/suppl/GSE63525_HUVEC_combined_30.hic.

[40] S. S. P. Rao et al. *K562 HiC-matrix*. URL: ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE63nnn/GSE63525/suppl/GSE63525%5FK562%5Fcombined%5F30%2Ehic.

[41] S. S. P. Rao et al. *NHEK HiC-matrix*. URL: ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE63nnn/GSE63525/suppl/GSE63525_NHEK_combined_30.hic.

[42] Ralf Krauth. *masterproject github*. 2020. URL: https://github.com/MasterprojectRK/HiCPrediction (visited on 05/05/2020).

[43] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. SciPy, 2010. DOI: 10.25080/majora-92bf1922-00a. URL: https://pandas.pydata.org/ (visited on 05/12/2020).

[44] Pauli Virtanen et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3 (Feb. 2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[45] The SciPy community. *Multi-dimensional image processing (scipy.ndimage)*. Dec. 19, 2019. URL: https://docs.scipy.org/doc/scipy/reference/ndimage.html (visited on 05/12/2020).

[46] Fidel Ramírez et al. "deepTools2: a next generation web server for deep-sequencing data analysis". In: *Nucleic Acids Research* 44.W1 (Apr. 2016), W160–W165. DOI: 10.1093/nar/gkw257.

[47] James Bergstra et al. “Algorithms for Hyper-Parameter Optimization”. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS’11. Granada, Spain: Curran Associates Inc., 2011, pp. 2546–2554. ISBN: 9781618395993.

[48] Shilu Zhang et al. *In silico prediction of high-resolution Hi-C interaction matrices (part I)*. 2019. DOI: 10.5281/ZENODO.3525431.

[49] Shilu Zhang et al. *In silico prediction of high-resolution Hi-C interaction matrices (part II)*. 2019. DOI: 10.5281/ZENODO.3525510.

[50] Shilu Zhang et al. *In silico prediction of high-resolution Hi-C interaction matrices (part III)*. 2019. DOI: 10.5281/ZENODO.3525514.

[51] Omar Elharrouss et al. “Image Inpainting: A Review”. In: *Neural Processing Letters* 51.2 (Dec. 2019), pp. 2007–2028. DOI: 10.1007/s11063-019-10163-0.

[52] Imren Dinc et al. “Evaluation of normalization and PCA on the performance of classifiers for protein crystallization images”. In: *IEEE SOUTHEASTCON 2014*. IEEE, Mar. 2014. DOI: 10.1109/secon.2014.6950744.