Albert-Ludwigs University Freiburg
Department of Computer Science
Bioinformatics Group

Master Thesis

Predicting Hi-C contact matrices using machine learning approaches

Author:
Ralf Krauth

Examiner:
Prof. Dr. Rolf Backofen

Second Examiner:
Prof. Dr. Ralf Gilsbach

Advisors:
Anup Kumar, Joachim Wolff

Submission date:
20.04.2021

Abstract

Harhar!

Zusammenfassung

Hohoho!

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus anderen Werken entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass die eingereichte Masterarbeit weder vollständig, noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens war oder ist.

Bad Krozingen, den 31. März 2021

Ralf Krauth

Contents

1 Introduction

In recent years, the three-dimensional organization of DNA has been shown to be a key factor for important processes in molecular biology. However, even with the most advanced experimental methods, it remains comparatively expensive to determine the spatial folding of DNA directly, so that current knowledge of three-dimensional DNA organization is still sketchy. In the last five years, several methods have thus been proposed to improve on this situation by determining DNA-DNA interactions *in-silico*. All of these are using certain types of experimental data which is easier to obtain than spatial data, but correlates with 3D chromatin structure in certain ways. However, most current in-silico approaches have disadvantages and shortcomings, and the current thesis attempts to improve on these.

1.1 Spatial structure of DNA

In the late 1970s, Watson and Crick discovered the now well-known double helix structure of DNA molecules [1]. However, this is not the only relevant spatial structure of chromatin. At larger scales, DNA molecules can be wound around certain proteins, so-called histones, forming DNA-protein complexes named nucleosomes. Several of these can further be compacted into fibers, which in turn can be "supercoiled" into the also well-known chromosomes, fig. 1.

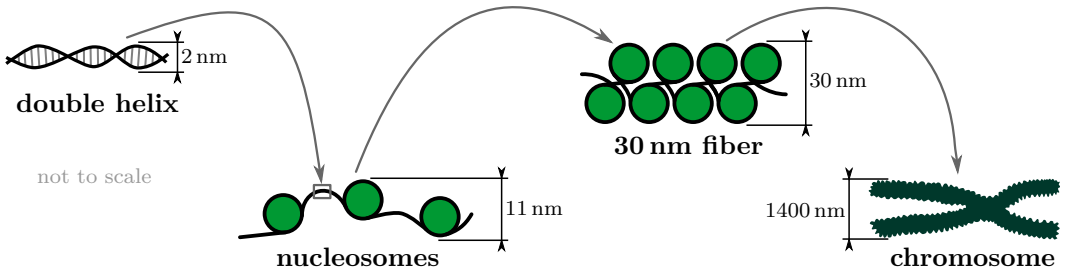


Figure 1: spatial chromatin structures (simplified, cf. [2])

While the spatial structure of chromatin outlined above brings along compaction, e. g. to make chromosomes fit into eukaryotic cell nuclei (fig. 1 from left to right), it also has other functional implications. One well known effect of spatial structure is the regulation of gene transcription by establishing or releasing contacts between certain DNA-regions, the so-called gene enhancers and promoters [3, 4]. Since enhancers can be up to a million basepairs away from the corresponding promoters, the two can usually only interact by means of spatial DNA structure. Further effects of spatial structure are still under investigation; two recent studies have for example found dependencies between chromatin conformation and cell differentiation in Drosophila Melanogaster [5] and investigated spatial structure dynamics during phase transitions in murine cells [6].

While the driving mechanisms behind the formation of spatial chromatin structures are partially still under research, too, certain proteins like CTCF or modified histones are already well-known to mediate or prevent DNA-DNA contacts [7, 8, 9]. This relationship can partially be exploited to predict 3D DNA conformation from the 1D binding sites of such proteins, as will be shown below.

In the last two decades, DNA-sequencing-based techniques have increasingly been utilized to capture the spatial structure of DNA experimentally. The method of choice within this thesis is called Hi-C and shall be explained in the following section.

1.2 The Hi-C process for determining spatial DNA structure

The Hi-C process is an elaborate biochemical procedure for investigating the spatial structure of DNA by detecting DNA-DNA interactions within and across chromosomes. The original Hi-C workflow has been developed by Lieberman-Aiden et al. in 2009 [10] and is depicted in simplified form in fig. 2.

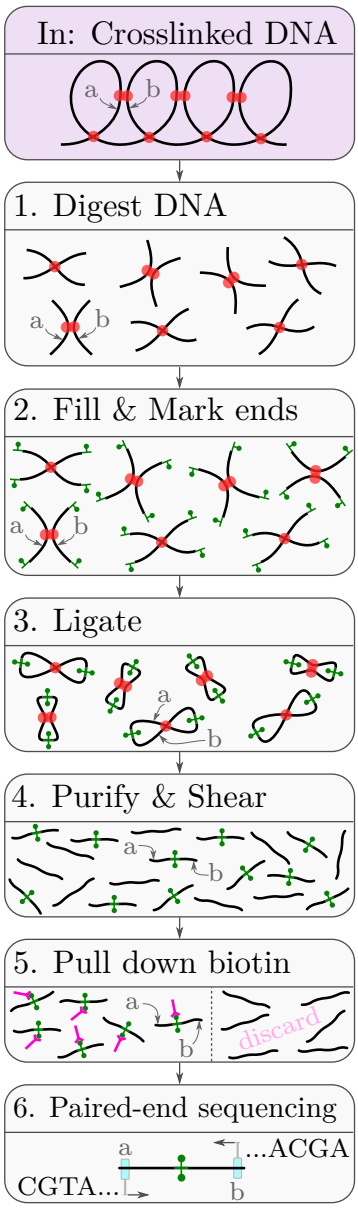


Figure 2: Hi-C lab process

The typical input (In) to Hi-C consists of several millions of cells, which are treated chemically to fix existing DNA-DNA contacts, commonly using formaldehyde, before they are lysed. Next, the DNA is extracted and cut into fragments by certain restriction enzymes (1), usually HindIII or DpnII, and the cut ends are repaired with nucleotides, some of which are marked by biotin (2). The free ends are then joined (3) under conditions which prefer ligations among open ends over ligations between different fragments. Originally, such conditions were achieved by high dilution of the fragments in solvents, but especially this part of the protocol has been replaced by more efficient methods in later works [11, 12]. The ligated fragments are then purified and cut into shorter sequences, some of which contain biotinylated nucleotides and some not (4). The fragments of interest – the ones containing biotinylated nucleotides – are then selected by pulling down biotin (5), for example using magnetic tags, and subjected to paired-end DNA-sequencing (6). In the end, the output of the Hi-C lab process is a large number of short genomic "reads", which are subsequently processed in the bioinformatics part of the Hi-C protocol outlined in the following section.

On the software side of the protocol, the reads first need to be mapped to the corresponding reference genome. Here, only reads are kept where the "left" sequence (6)(a) uniquely maps to a different region of the reference genome than the "right" sequence (6)(b). These so-called chimeric reads are subjected to quality control, and those passing are counted as an interaction between the two genomic positions (a) and (b) to which the two ends belong. However, at reasonable read coverages, interactions cannot be counted per base pair. Instead, the reference genome is split into equally sized bins (or regions), and the reads are counted for those bins where they belong. Common bin size values b include $b \in \{1, 5, 10, 25, 50, 100, 1000\}$ kbp.

The final outcome of a Hi-C experiment is then a (sparse) square matrix M, henceforth referred to as "Hi-C matrix", which records the interaction count for all possible pairs of regions in the

reference genome. The individual elements $m_{i,j}$ of the matrices are counts of interactions between the bins with indices $i, j \in \{0, 1, ...\}$, where each bin index i and j uniquely maps to a genomic region with defined start- and end position. For example, if region index i corresponded to "chr1, 25000...49999" and index j corresponded to "chr1, 250000...274999", then a matrix entry $m_{i,j} = 22$ would mean that 22 interactions have experimentally been measured between the two mentioned genomic positions. Because interactions do not have a "direction", Hi-C matrices are always symmetric and it thus holds that $m_{i,j} = m_{j,i}$.

In the bioinformatics part of the Hi-C protocol, often only a small fraction of all reads fulfill the selection criteria outlined above, for example due to reads not being chimeric or uniquely mappable. This makes Hi-C a comparatively inefficient, slow and thus expensive process. For example, the well-known dataset by Rao et al. [11] with matrix bin sizes down to 1 kbp, required several *billions* of reads being made.

Parts of fig. 2 and the process description above have been adapted from the preceding study project [13].

1.3 The ChIP-seq process for determining protein binding sites

The ChIP-seq process is a combination of Chromatin-Immunoprecipitation (ChIP) and DNA-sequencing, designed for investigating DNA-protein interactions [14, 15]. As with Hi-C, the input consists of a sufficient number of cells, which are first treated with formaldehyde to fix present proteins to DNA, fig. 3 (In). The cells are then lysed and the DNA-protein structure is extracted and cut into fragments, usually by sonication (1). Next, specific antibodies are added, designed to bind only to a certain protein of interest (2). 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 (3). The proteins and antibodies are then removed (4), the DNA is purified and finally sequenced (5). Typically, a control experiment is performed together with the ChIP-seq process, which comprises all steps described above, except the immunoprecipitation (2),(3).

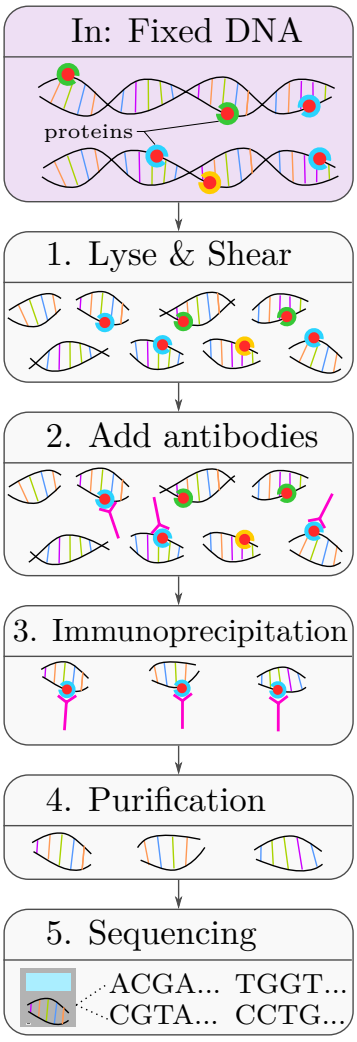


Figure 3: ChIP-seq lab process

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.

On the software side of the process, the reads are filtered for quality and mapped to an appropriate reference genome. The number of mapped reads per genomic position can then simply be counted. It is common to process reads from the control experiment in the same way as reads from the ChIP-seq experiment

and then use special software to call "peaks", i. e. protein binding sites, at those genomic positions where the read count from ChIP-seq is statistically significant compared to the read count from the control group.

For the thesis at hand, the read counts from the ChIP-seq experiment have been used directly without calling peaks, because peak data was found to be too sparse for the machine learning approach used in the study project [13].

Parts of fig. 3 and the process description above have been adapted from the study project [13].

1.4 Motivation and goal of the thesis

Due to the high effort, Hi-C experiments have been published for comparatively few organisms and cell lines so far. For example, as of January 2021, Hi-C assays were available for only 26 of more than 150 human cell lines listed in the Encyclopedia of DNA elements [16, 17], table 1, and these have been produced by only five university labs.

cell line	TF ChIP-seq	Histone ChIP-seq	Dnase-seq	Hi-C
K562	680	20	10	1
HepG2	668	15	3	1
A549	251	87	14	6
HEK293	231	6		
GM12878	211	15	3	7
MCF-7	155	18	8	
H1	88	55	3	
HeLa-S3	77	15	4	1
SK-N-SH	62	21	2	
IMR-90	16	34	2	2
HCT116	27	17	1	8
H9		35	7	
GM23338	15	13	1	
Ishikawa	25		4	
HEK293T	17		1	
GM23248	2	13	1	1

Table 1: availability of selected assays in ENCODE (extract)

However, investigations and experiments with the available Hi-C data have shown correlations between the spatial DNA structure and the binding sites of certain transcription factors and histone modifications [11, 18]. Since these binding sites can be detected with the less costly ChIP-seq method, a computational method able to predict contact matrices from DNA-protein bindings *in silico* would be very helpful.

The goal of this master thesis is thus to predict missing Hi-C data from existing ChIP-seq data, making use of the known correlations between the binding sites of transcription factors and histone modifications on the one hand and Hi-C interaction counts on the other hand. Because this is a wide field and lead time for a master thesis is limited, the present work will focus on machine learning techniques, which have proven a good choice for exploiting complex patterns and relationships, especially when it comes to the synthesis of 2D data from various kinds of input data [19]. More specifically, the goal of this thesis is to develop an easy-to-use machine learning approach for predicting Hi-C interaction matrices from ChIP-seq data, using standard in- and output formats with minimal pre- and post-processing.

To underscore the usefulness of such an approach, table 1 to the side shows the first 16 human cell lines in ENCODE, sorted by total number of selected assays available. Although not necessarily all available experiments to date are included in ENCODE – for example, the experiments by Dixon et al. [9] for H1 stem cells are not listed – it is obvious that the public availability of ChIP-seq data is commonly much better than the one of Hi-C data, even for otherwise well investigated cell lines like HEK293 and MCF-7.

2 Related work

In the last five years, several approaches have been presented to determine DNA-DNA interactions *in silico*, using existing data from various experiments. Section 2.1 gives an overview about these methods. Furthermore, some methods originally developed for image synthesis and similar tasks in computer vision might also be useful in the field of Hi-C matrix generation and are thus summarized in subsection 2.2. The section is then concluded by a short discussion of the existing work.

2.1 Methods for predicting DNA-DNA interactions and contact matrices

As of 2020, there is quite a body of existing work in the field of predicting DNA-DNA interactions, using various approaches and different types of input data.

Two conceptually similar methods have been proposed by Brackley et al. in 2016 and MacPherson et al. in 2018 [20, 21]. In both approaches, DNA is modeled as a "beads-on-a-string" polymer, and simulation techniques are employed to find energy-optimal spatial structures of these polymers. Apart from constraints derived from the molecule's DNA sequence itself, the models also consider spatial contact constraints derived from ChIP-seq experiments of chromatin factors which are known to mediate such DNA-DNA contacts. The interaction matrices derived from the simulations look interesting, but the paper from Brackley et al. [20] is unfortunately lacking a comparison with "true" experimentally measured Hi-C matrices, and the results from MacPherson et al. [21] seem inferior to most other ones presented in this section.

Another simulation-based method has been developed by di Pierro et al. in 2017 [22] and later extended by Qi and Zhang [23]. In both cases, a convolutional neural network (CNN) is trained to learn different "open" and "closed" chromatin states from 11 chromatin factors, and the predicted chromatin states are then taken as constraints for beads-on-a-string models. The difference between [22] and [23] lies mainly in the number of states considered and the simulation methods applied; the results are mathematically convincing in both cases.

A further approach using chromatin states is due to Farrè and Emberly [24]. Here, the conditional probability of two genomic regions being in contact, given their distance and the chromatin state around them, is estimated using Bayes' rule. In this case, the chromatin state – reduced to active or inactive – is derived from DNA adenine methyltransferase identification (DamID) signals of 53 chromatin factors using probabilistic methods [25]. The conditional probabilities on the right side of Bayes' rule are either computed from training data or estimated with different probabilistic approaches, too. While the predicted contact matrices do not look like real Hi-C matrices, highly interacting regions are still often well identifiable with this approach.

Three further approaches in the field make use of machine learning in form of random forests. *3DEpiLoop* by Bkhetan and Plewczynski and *Lollipop* by Kai et al., both from 2018, use a random forest classifier to predict DNA loops, but differ in input data and preprocessing [26, 27]. While 3DEpiLoop is using only ChIP-seq data of histone modifications and transcription factors [26], Lollipop additionally takes ChIA-PET-, RNA-seq- and DNase-seq-data, CTCF motif orientation and loop length as inputs [27]. Both approaches show good coincidence of predicted loops with experimental data, but their output is binary and rather sparse. Contrary to these two, the third random-forest-based approach, *HiC-Reg* by Zhang et al. from 2019,

allows predicting real-valued Hi-C interaction matrices directly [28]. To this end, it employs random forest regression to predict interactions between all pairs of genomic regions within a certain distance. For each pair of regions and the genomic window enclosing them, ChIP-seq- and DNase-seq data of 14 chromatin features are taken as decision criteria for the random forest, along with genomic distance of the pair. The published results for five human cell lines look visually good in terms of Hi-C matrix plots, while the reported correlations between predicted and true matrices are modest.

Another recent method which investigated decision-tree-based algorithms is due to Martens et al. (2020) [29]. Here, gradient boosted decision trees, logistic regression and neural networks were used to predict highly interacting chromatin regions and TAD domain boundaries from histone modifications and CTCF ChIP-seq data. In this setup, the neural network approach yielded the best, overall acceptable results of the three approaches, but again in form of a binary classifications. A neural-network approach with comparable input data, but without the restriction to binary classifications has been presented by Farrè et al. in 2018 [30]. Here, a one-dimensional convolutional filter is used to convert ChIP-seq data of 50 chromatin factors from a certain region of the genome into a one-dimensional chromatin vector, which is then processed by a dense neural network (DNN) to predict Hi-C submatrices. Using a sliding window approach, the method by Farré et al. allows predicting complete, real-valued Hi-C interaction matrices, which resemble the general structure of experimentally derived matrices quite well.

While the approaches discussed so far have either modeled DNA as a "beads-on-a-string" polymer or not used the actual DNA sequence explicitly at all, there are also several machine-learning approaches which directly consider DNA sequence without a need for polymer modeling. In 2019, Singh et al. presented *SPEID* [31], an approach to predict promoter-enhancer interactions from DNA sequence, using a combination of CNNs, a recurrent network (LSTM) and a DNN. The results match well with experimental data, but are limited to promoter and enhancer loci by design, disallowing predictions of complete Hi-C contact matrices. Other researchers have tried to design similar methods without such limitations. The work by Peng from 2017 [32] is an extension of SPEID, based on a 2016 preprint [33], additionally taking into account a "middle sequence" between enhancer- and promoter sequences, CTCF motif counts within the sequences and genomic distance between two sequence snippets. However, the network lacks generalization, i. e. the results are only good in training regions [32, figs. 4, 5]. A conceptually similar method to the one by Peng [32], but with a different neural network design has been presented by Schreiber et al., also in 2017, named *Rambutan* [34]. It accepts DNA sequence, DNase-seq data and distance between two genomic loci as inputs and then uses a combination of CNNs and a DNN to predict whether the given two loci interact or not. Unfortunately, it is difficult to decide whether the results of Schreiber et al. are useful for the task at hand, since the evaluation is done only by statistical means and no actual Hi-C matrices have been published. The original paper [34] also contains a known error and seems not to have appeared in a peer-reviewed journal in improved form yet. A probably more promising method working on DNA sequence, *Akita*, has been published by Fudenberg et al. in 2020 [35]. It is based on two rather involved convolutional neural networks. While the first one, "trunk", processes one-dimensional, one-hot encoded DNA sequence input through convolutional filters, the second one, "head", converts one-dimensional representations to 2D, further processes the data with convolutional filters and enforces symmetry. Although Fudenberg et al. initially seemed to focus on determining the influence of DNA modifications on spatial structure [36], predicting complete Hi-C matrices is an integral part of their work, and a large number of images of Hi-C matrices from the test set has

been published alongside the article. The predicted matrices often hardly look like experimental Hi-C matrices, but mostly still indicate highly interacting regions quite well.

A further method by Schwessinger et al. [37] also makes use of DNA sequence and additional epigenetic data for its predictions, but is conceptually different from the ones presented so far. Here, ChIP-seq tracks are initially used to train a CNN on the relationship between sequence and the corresponding chromatin factors. The weights of this first network are then used to seed another convolutional neural network, which is responsible for predicting DNA-DNA interactions from DNA sequence. In this case, the results were blurry, but generally in good accordance with experimental Hi-C data.

Yet another machine-learning concept based on sequence data, *Samarth*, has been proposed by Nikumbh and Pfeifer in 2017 [38]. Here, a support vector machine is trained on 5C data, using a specific representation for DNA sequence called oligomer distance histograms. The results showed acceptable consensus with experimental Hi-C data and allowed some interesting conclusions about the importance of certain k-mers for DNA folding. However, the applicability for the task at hand is hard to assess, because none of the predicted matrices used for statistical evaluation has been published alongside the paper.

2.2 Image synthesis techniques from computer vision

With the advent of deep learning, both the number of opportunities and the demand for using machine learning techniques in image synthesis tasks have risen, as recently summarized by Tsirikoglou et al. [19]. Since Hi-C contact matrices can be seen as square grayscale images, such techniques can also be relevant for this thesis.

Probably one of the first applications of computer vision methods for Hi-C matrices was presented by Liu and Wang in 2019 [39]. Here, established image super-resolution networks – mostly deep convolutional neural networks – have been modified to enhance low-resolution Hi-C matrices.

Another technique from computer vision that has been transferred to Hi-C matrices are conditional generative adversarial networks (cGANs), invented by Goodfellow, Mirza and colleagues in 2014 [40, 41]. Again, several works have employed this comparatively new and involved method to increase the resolution of given Hi-C matrices, including the ones by Liu and colleagues [42], Hong et al. [43] and Dimmick et al. [44]. In general, all three works feature the typical cGAN setup with two competitive networks – a *generator*, which is trained to produce realistic high-resolution Hi-C matrices from random noise and the low-resolution matrices (as conditional input), and a *discriminator*, which tries to distinguish real Hi-C matrices from generated ones. In this setting, the discriminator serves as a "critique", an additional loss function, for the generator, which has been shown empirically to be beneficial in many applications. While the convolutional building blocks for the discriminators and the residual building blocks for the generators are conceptually similar in all cases, the three works differ in the number of building blocks and the activation functions applied within the blocks. Furthermore, Dimmick et al. and Hong et al. include additional loss functions beyond standard generator- and discriminator losses. The method by Dimmick et al. outperformed the others for most test cases, but it is also the most elaborated.

2.3 Discussion of existing work

Independent of chosen techniques, several of the methods shown above only allow predictions restricted to certain loci (e. g. promoters and enhancers) or yield binary predictions in the sense of "interaction" or "no interaction" between certain loci [26, 27, 29, 31]. These methods are thus not directly suitable for the task at hand, but would require further development.

The chromatin-modeling based approaches [20, 21, 22, 23] allow indirect derivation of Hi-C matrices by performing sufficient simulation passes and estimating contact counts from the resulting model ensembles. Depending on the chromatin model choice – which seems not straightforward [45, 46] – and the number of constraints involved, the required computations can be expensive. However, the results still seem slightly inferior compared to other methods mentioned in section 2.1 – maybe because only modeled constraints can be considered, but not all constraints for chromatin conformation are known or can be modeled yet.

Three of the DNA-sequence based methods mentioned above also allow direct prediction of Hi-C matrices [34, 35, 37]. At first look, it is surprising that learning spatial structure from DNA-sequence actually works, because 3D conformation can vary considerably for different cell lines of the same organism, which all share the same DNA sequence. On the other hand, the chosen artificial networks might be able to figure out binding sites of relevant proteins from DNA sequence, which *do* have a correlation with spatial structure. This is especially true for the method by Schwessinger [37], where the network is seeded by training on exactly such binding sites. While the methods by Schreiber and Schwessinger use secondary inputs and therefore can – at least partially – adapt to cell lines sharing the same DNA, the method by Fudenberg focuses on DNA and is thus expected to produce the same outputs for all cell lines of a given organism. All three methods require comparatively deep networks which are expensive to train. The fourth sequence-based method by Nikumbh et al. [38] is using a support vector machine, which is generally easier to train, but the adaptability to different cell lines is expected to remain problematic due to the chosen concept.

The random-forest-based method by Zhang et al. [28] is directly targeted at the goal of this thesis, since it directly predicts Hi-C matrices, is not limited to certain loci and can adapt to different cell lines using corresponding ChIP-seq data. This approach has extensively been investigated in two previous study projects [13, 47].

The dense neural network approach by Farrè et al. [30] is also compatible with the goals of this thesis. The published results are visually and statistically convincing, and both the presented network and the training process offer opportunities for amendments, which will be explored in section 3.1.

Since all of the image synthesis methods presented above in section 2.2 require existing Hi-C matrices for training *and* prediction, none of them is directly suitable for the task at hand. However, some of the concepts can still be used to develop novel approaches, which will be discussed in section 3.2.

3 Advancing predictions of Hi-C interaction matrices

In the following subsections, two conceptually different approaches towards the goal of the thesis, predicting Hi-C matrices from ChIP-seq data, will be explored. While the first approach is a dense neural network based on work by Farré et al. [30], the second is a novel method based on conditional generative adversarial networks (cGANs).

3.1 Dense Neural Network approach

In their 2018 paper [30], Pau Farré, Alexandre Heurtau, Olivier Cuvier and Eldon Emberly proposed a combination of a 1D convolutional filter with a three-layer dense neural network which already fulfills most goals of this thesis with some exceptions regarding data formats and preprocessing, cf. section 2.1 and 2.3. This thesis attempts to build on the success of their method by extending the comparatively simple neural network in various ways, changing the learning process and the loss functions in use. As a start, the basic network has been rebuilt and used on well-known data from human cell lines GM12878 and K562, cf. section 4.1.

3.1.1 Basic network setup

The basic network setup taken over from Farré et al. [30] is shown in simplified form in fig. 4, see section 4.7.1 for technical details.

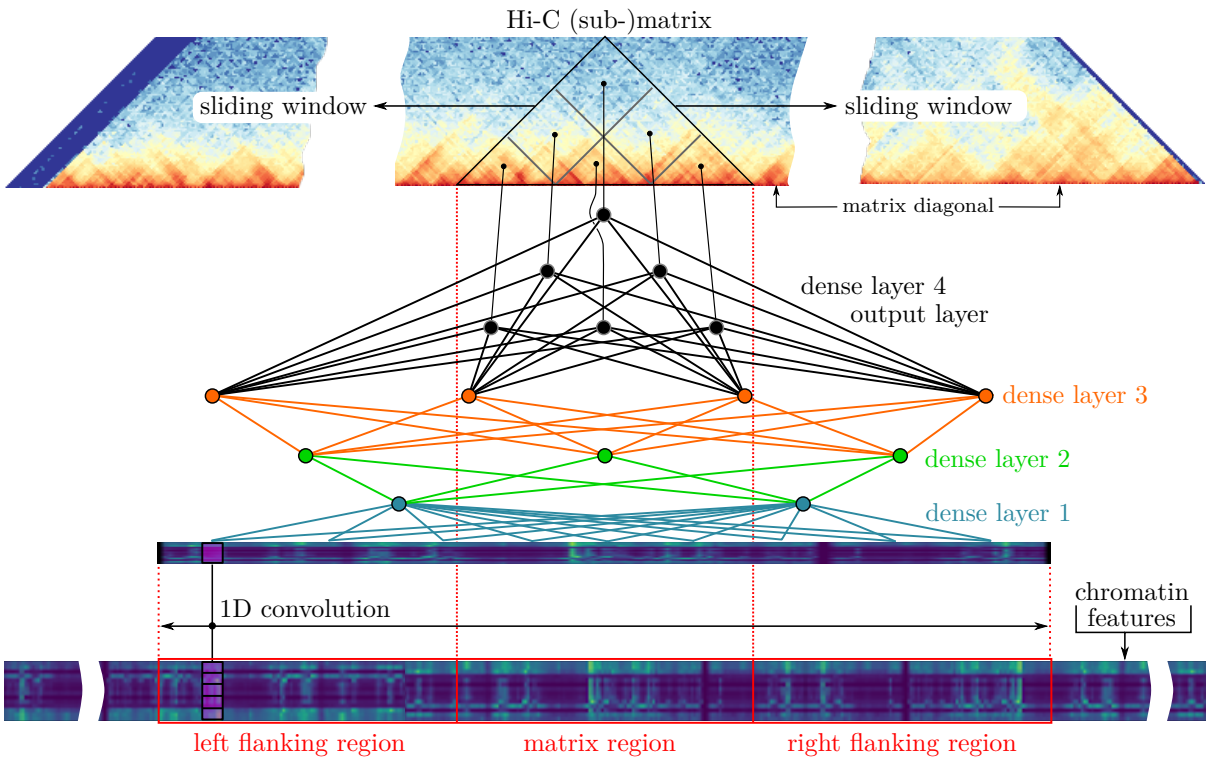


Figure 4: Principle of basic dense neural network

Since most organisms feature only a small number of chromosomes, it is generally infeasible to learn Hi-C matrices for full chromosomes at once due to a lack of training data. Instead, the method by Farré et al. employs a sliding window approach and learns from submatrices of a certain size, which allows for a reasonable number of training samples. The target matrices are thus taken as overlapping submatrices of size (w, w) with fixed window size w, centered at the diagonal of the original Hi-C matrices, fig. 4 top. The ChIP-seq data of n chromatin features is taken as $(3w, n)$-subarray of the original array, whereby the middle w bins are aligned with the position of the submatrix, the first w bins correspond to the left flanking region and the last w bins correspond to the right flanking region of the current submatrix region, fig. 4 bottom. Samples are then obtained from the input data by sliding the input windows along the diagonal of the target Hi-C matrix with step size one. For predictions, overlapping samples can be re-assembled into complete matrices without difficulties by taking the mean prediction for each genomic position. Further technical details of the sample generation process are given in section 4.2.

Within the network, a 1D convolutional filter compresses the $(3w, n)$-shaped input arrays to 1D vectors of length $3w$, and four dense layers further process the compressed input, fig. 4 middle. The number of neurons in the last dense layer corresponds to the number of bins in the upper triangular part of the target submatrix, i. e. it consists of $(w \cdot (w + 1))/2$ neurons, fig. 4 top, exploiting the symmetry of Hi-C matrices. For implementation details, please refer to section 4.7.1.

Training of the network is performed by minimizing the mean squared error of the predicted matrix versus the target Hi-C matrix using stochastic gradient descent, cf. section 4.7.1.

Farré et al. propose a window size of $w = 80$ at $b_{feat} = b_{mat} = 10\,\text{kbp}$, which comes close to the maximum bin distance available in the Hi-C data by Schuettengruber et al. [48] used in their publication [30]. However, larger bin sizes of $b_{feat} = b_{mat} = 25\,\text{kbp}$ were found beneficial for most of the data used throughout this master thesis. Additionally, larger bin sizes allow for a higher coverage of the target matrix at the same window size, because the window size is specified in bins, and obviously $10\,w < 25\,w$. Prediction results from the initial network for windowsize $w = 80$ and both bin size 10 and 25 kbp are shown in section 5.1.1.

The network layout shown above is quite simple, and immediately offers some opportunities for expansion, partially already proposed in the original paper [30]. These will be explored below.

3.1.2 Modifying the convolutional part of the network

One starting point for modifying the neural network is its convolutional part.

With only a single 1D convolutional filter in one layer, the network was expected to have difficulties capturing complex relationships between Hi-C interaction counts and more than one of the chromatin features. For this reason, an extended "longer" network was created, comprising three 1D convolutional filter layers with 16, 8 and 4 filters, respectively, replacing the single 1D-convolution in fig. 4, lower left, cf. section 4.7.2. This is still a comparatively low number of layers and filters, but the choice seemed justified in order to avoid overfitting to the low-dimensional input. The results are shown in section 5.1.2, especially figures 27 and 28.

Next, a "wider" network was created, featuring the same setup as the basic network except the width of the filter kernel, which was set to 4 instead of 1. The idea here was to allow the network to capture correlations between Hi-C interaction counts and chromatin features which span more than one bin. The actual number has been kept low, since at bin size $b = 25\,\text{kbp}$, 4 bins already correspond to 100 kbp. However, the results were not as expected, cf. section 5.1.2, especially figures 25 and 26. Of course, increasing filter width and using more filters can also be combined, hopefully allowing the "wider-longer" network to capture both correlations spanning more than one bin and more than one chromatin feature. The results for this combined approach are shown in section 5.1.2, especially figures 29 and 30.

Another approach to potentially improve the predictions that goes somewhat into the direction of the "wider" network has been proposed, but not implemented by Farré et al. in their paper [30]. Read counts from ChIP-seq experiments can usually be binned at smaller bin sizes than Hi-C data due to the nature of the processes. This can be exploited to capture finer details in the ChIP-seq data without a need for higher (training-)matrix resolutions. To this end, the initial network can be generalized by binning the ChIP-seq data at k times the bin size of the matrices, whereby $k \in \mathbb{N}^{\geq 1}$, cf. section 4.4 for the technical details. This yields an input data size of $(k \cdot 3w, n)$, which is then again compressed to a vector of length $3w$ by a 1D convolutional filter with kernel size n and strides k. To match given matrix bin sizes, $k = 5$ was chosen for the thesis at hand, and the results for bin sizes $b_{mat} = 25\,\text{kbp}$, $b_{feat} = 5\,\text{kbp}$ are shown in section 5.1.2, especially figures 31 and 32.

Since hardly any improvements were recorded for the modifications of the convolutional network part described above, advanced loss functions were investigated next.

3.1.3 Using a combination of MSE-, TV- and perceptual loss

In image regression tasks, optimizing for mean squared error (MSE) is known to produce blurry images, because MSE is treating each image pixel individually, ignoring spatial proximity and structures in images [49, 50]. And indeed, both the predictions from the initial and the extended network according to sections 3.1.1 and 3.1.2 suffered from blurriness. To improve on this, investigations with modified loss functions were made.

It has been shown that loss functions based on the (multiscale-)structural similarity index (SSIM) [51]. can outperform mean squared error (L2) and mean absolute error (L1) in image regression tasks. While Zhao et al. used a combination of L1- and multiscale SSIM loss [52], Lu proposed a custom level-weighted SSIM loss [50]. The results were better than with L1- or L2-loss alone, but sometimes not much – depending on the machine learning model in use. Owing to difficulties in finding suitable parameters, SSIM-based losses were not considered for the thesis.

Another type of loss function used to produce sharp images is the so-called perceptual- or perception loss. Here, the idea is to use a pre-trained perception network to determine structures in images and then compute for example L1- or L2-loss on these structures instead of images, which has led to visually better predictions in some applications. The process is shown in more detail in fig. 5.

As usual, training samples are fed into the network to be trained, and an output – here, the predicted Hi-C submatrix – is generated, fig. 5, orange path. However, the loss function is then

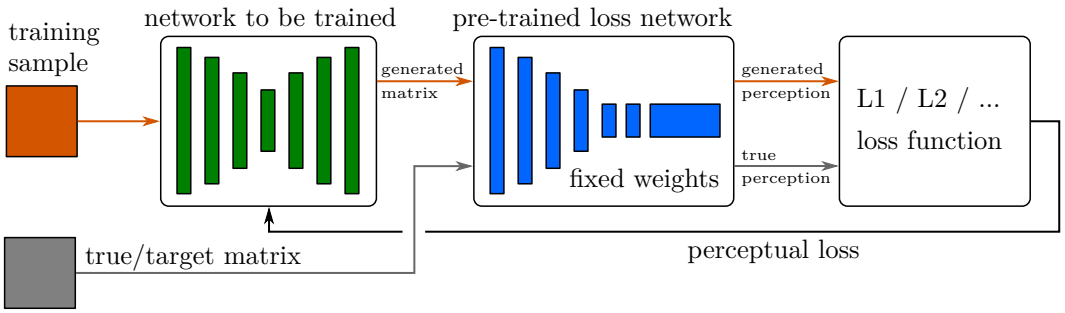


Figure 5: Perceptual loss

not computed on the predicted versus the true matrix. Instead, the activations of a selected layer in the perception network are generated for both the predicted Hi-C submatrix (orange path) and the true Hi-C submatrix (gray path), and the loss function is computed on these activations (or perceptions). The optimization of the target network's trainable weights can then be performed as usual, for example with stochastic gradient descent and backpropagation, simply keeping the weights of the perception network constant while optimizing the trainable weights for minimum perceptual loss.

Often, complex image classification networks like VGG-16 [53] are taken as loss network, e. g. in the well-known style-transfer network by Johnson et al. [54], because certain layers in these networks are known to have a correspondence with relevant structures in images. It is possible to use multiple perceptions in a joint loss function [54] and combine perceptual losses with other losses, as shown below.

To check whether the given learning task benefits from using a perceptual loss, a custom combined loss function was generated, consisting of mean squared error L_{MSE} between true- and predicted matrices, perceptual loss L_{VGG} based on the VGG-16 network and total variation loss L_{TV} on predicted matrices to reduce noise in the output while preserving edges [55]. This choice was inspired by the custom loss function used by Hong et al. in their successful Hi-C super-resolution network *DeepHiC* [43], which is otherwise not similar to the network used here. In short, the combined loss function L is shown in eq. (1). Here, the λ are individual loss weights, see section 4.7.3 for details.

$$L_{combined} = \lambda_{MSE}\, L_{MSE} + \lambda_{VGG}\, L_{VGG} + \lambda_{TV}\, L_{TV} \tag{1}$$

Unfortunately, there is no straightforward way for determining the optimal parameters λ, and an exhaustive parameter search was infeasible due to the computation time requirements of about 4:30 min per epoch. Therefore, only few runs were conducted with different sets of parameters, and the results for $\lambda_{MSE} = 0.8999, \lambda_{VGG} = 0.1, \lambda_{TV} = 0.0001$, which should not be considered optimal, are shown in section 5.1.3.

3.1.4 Using a TAD-based loss function

Looking at the results obtained from the networks so far, see XXX, it seemed that highly interacting regions, especially topologically interacting domains (TADs), were not well predicted

and either absent in the matrix plots or blurred. Assuming availability of a TAD scoring function $tad(\mathbf{z}_{pred})$, where $\mathbf{z}_{pred}$ is a predicted submatrix, this might be improved by directly optimizing a loss function as shown in eq. (2).

$$L_{combined} = \lambda_{MSE}\, L_{MSE} + \lambda_{TAD}\, (tad(\mathbf{z}_{true}) - tad(\mathbf{z}_{pred}))^2 \tag{2}$$

However, this approach suffers from several restrictions. First and foremost, there seems no consensus on the exact definition of TADs, and no less than 22 algorithms for TAD detection existed as of 2018 [56, 57]. Additionally, many of these algorithms have several tuning parameters, are notoriously parameter-dependent and may not even yield any results if parametrized in an unfavorable way [57]. A further restriction results from the context – since the loss function needs to be optimized, one needs to be able to compute gradients of it with respect to the network's weights. Due to their complexity, this is generally very difficult to implement in a computationally efficient way for all known TAD calling algorithms.

To overcome the limitations, a novel loss function based on TAD scores [58] was developed. Figure 6 exemplarily shows its basic idea for a $(16, 16)$-shaped submatrix with window size 4. First, the mean is taken from diamond-shaped (or rhombus-shaped) matrix cutouts along the

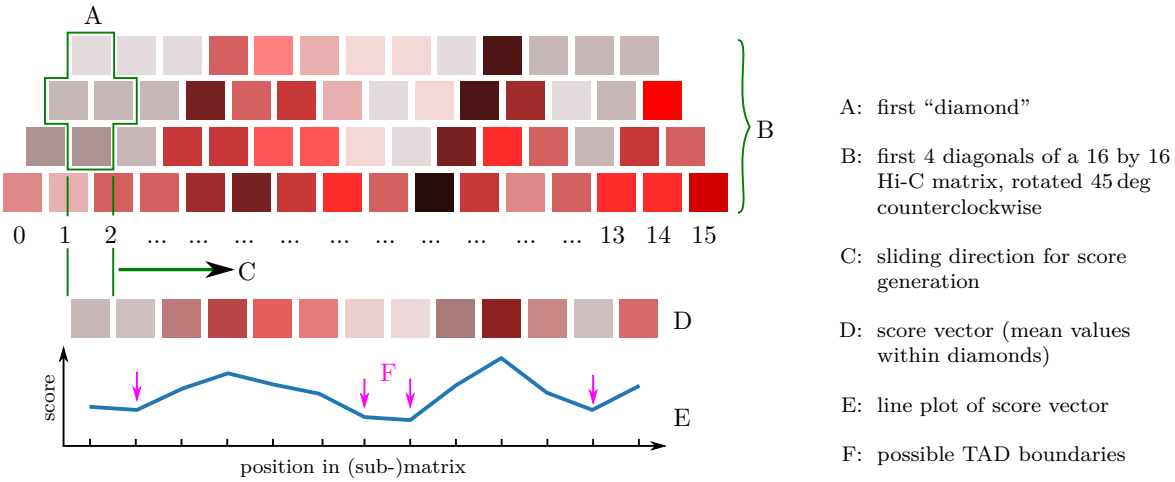


Figure 6: Score vector generation for TAD-based loss

diagonal, fig. 6 (A, C), and stored in a score vector, fig. 6 (D / E). The size of the diamonds, 2 in the figure, is configurable, but a reasonable balance with the submatrix size, i. e. the size of the sliding window w, and the expected size of TADs in the matrix must be maintained. Next, loss is computed by taking the mean squared error between the score vectors of the "real" submatrices and the "predicted" submatrices, eq. (2).

The idea behind the score-based approach is visualized in fig. 6 (E). Local minima, i. e. dents in the line plot of the score values, fig. 6 (F), often correspond with highly interacting regions in the matrix, since the mean of diamonds from *inside* TADs is normally significantly higher than the mean of diamonds *outside* TADs. Indeed, some TAD calling algorithms like *TopDom* [59] and *hicFindTADs* [60, W12f.] do compute insulation scores – usually for more than one diamond size – and then use diverse techniques to detect meaningful local minima in the score values. However, finding meaningful local minima in the given context is still computationally involved, so it was left to the network to make sense out of the score vectors. This way, the loss

function was well defined, efficiently computable and tensorflow standard functionality could be used to compute gradients with respect to weights.

For the thesis at hand, window size $w = 80$ and diamond size $ds = 12$ were used at bin sizes of 25 kbp. For technical details please refer to section 4.7.4.

3.1.5 Modifying bin size and window size

In the results presented so far, larger structures were often absent. In order to improve on this, two approaches were tried.

First, predictions were made at bin sizes $b_{feat} = b_{mat} = 50\,\text{kbp}$, keeping the window size at 80 bins, which then corresponded to 4 Mbp. The idea here was to capture predominantly larger structures further away from the diagonal and then investigate various methods to combine predictions at smaller and larger bin sizes. However, such a merging process was never actually developed, since the predictions at 50 kbp alone were not good enough, section 5.1.5.

Unfortunately, doubling the (matrix-)bin sizes from 25 to 50 kbp directly leads to a reduction in the number of available training samples by a factor of about two, if the window sizes are kept constant at 80 bins, see table 3. This might also be one of the causes why predictions at 50 kbp proved useless. For this reason, a second approach was made – using training samples at $b_{feat} = b_{mat} = 25\,\text{kbp},\ w = 80$ and $b_{feat} = b_{mat} = 50\,\text{kbp},\ w = 80$ at the same time. This is easily possible, since the neural network topology only depends on the window size *in bins* and the relation $k = b_{mat}/b_{feat}$ between the bin sizes of features and matrices. This approach obviously increases the number of training samples, with the idea of allowing the network itself to figure out how to combine predictions at smaller and larger bin sizes.

- use trapezoids, i.e. capped larger submatrices and flanking size smaller than window size
- rationale: larger window size without increasing training time too much

3.1.6 DNA sequence as an additional network input branch

- use DNA as an additional input
- rationale a): allow the network to figure out true binding sites in conjunction with cs data
- rationale b): given the success of pure DNA based methods, allow the network to find yet unknown sequence structure correlations
- probably not the most important subsection, leave it out in case of time problems

3.2 Hi-cGAN approach

Because none of the results from the dense neural network approach presented in section 5.1 were overly convincing, a second, widely unrelated approach was made.

Since their invention by Goodfellow, Mirza and colleagues [40, 41], Generative Adversarial Networks have become increasingly popular for image processing tasks of many kinds, and especially for image synthesis [61]. Among their strengths is image synthesis from textual descriptions [62, 63, 64, 65] – and from an abstract point of view, this task is not very different from the goal of the thesis at hand. Considering the chromatin features on the input side as a "description" of how the target Hi-C (sub-)matrices should look like, formulating the goal of the thesis as a cGAN-problem should be possible. In the following sections, such an approach will be explored.

3.2.1 General setup of the cGAN approach

Although numerous variants exist, conditional GANs generally comprise at least two neural networks – a generator $G(x, z)$, which tries to generate realistic outputs from its inputs x and random noise z, and a discriminator $D(x, y)$, which tries to discern true inputs y, e. g. experimentally derived Hi-C matrices, from generated inputs $G(x, z)$. The optimal weights for the networks can then be found by searching an equilibrium in a two-player minimax game: The generator tries to fool the discriminator, while the discriminator tries to detect the generator's fakes, see equations 3 and 4 [49], where $\mathbb{E}$ is the expected value for each "player".

$$L_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))] \tag{3}$$

$$G^* = arg\, min_G max_D\, L_{cGAN}(G, D) \tag{4}$$

Many different layouts and training processes for the Generator and Discriminator networks have been proposed. For the thesis at hand, the well-known *pix2pix* proposal by Isola et al. from 2017 was followed [49], amended by a convolutional embedding network for the chromatin features, which serve as the conditional input x here. The overall setup is shown in fig. 7 and will be explained in more detail below.

The generator architecture is based the well known U-Net architecture due to Ronneberger, Fischer and Brox [66], while the discriminator is implemented as a patchGAN-discriminator developed by Isola et al. especially for *pix2pix* [49]. This type of discriminator splits the images into patches and tries to decide which of them are real and which are fake, using convolutions. Note that *pix2pix* does not explicitly add noise z, but by design only relies on dropout layers for that purpose.

For the thesis at hand, few modifications were made to the actual *pix2pix*network. Since Hi-C matrices are symmetric by definition, symmetry of the outputs was enforced by adding additional layers to the network in the appropriate places, see section 4.8.1 for details. Furthermore, some layers in the generator and discriminator were made optional to allow processing smaller images of sizes 64×64 and 128×128 aside the 256×256-images of the original implementation, again refer to section 4.8.1 for details.

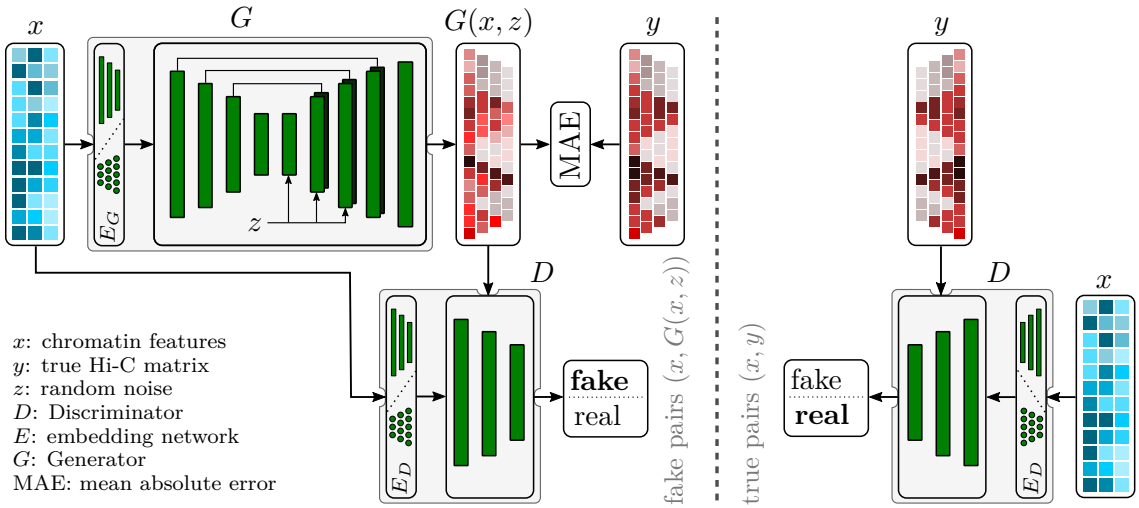


Figure 7: cGAN approach

For sample generation, the same sample generation method as for the dense neural network described above was employed, yet with a few minor adaptations, see section 4.3. However, since *pix2pix* operates on images, as its name already implies, a method to add the essentially one-dimensional chromatin feature data as conditional input "images" into the network was needed. Unfortunately, this task seems to be without examples in literature, so two different embedding approaches were made, which will be discussed below in sections 3.2.2 and 3.2.2. Both approaches are distantly related to the text-encoders used by text-to-image synthesis networks [67, 68], but much less sophisticated. However, a complex encoding is not needed here, since – contrary to image captions – the chromatin features are already in a format that can be processed directly by neural networks.

Many ways for training a cGAN exist, and achieving a stable, converging training process can be tricky. For the thesis at hand, the generator loss function proposed by Isola et al. [49] was used, amended by a TV loss function for noise reduction, eq. (5).

$$L_G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \lambda_{MAE} L_{MAE}(G(\mathbf{x}, \mathbf{z}), \mathbf{y}) + \lambda_{adv}\, L_d(\mathbf{1}, D(\mathbf{x}, G(\mathbf{x}, \mathbf{z})) + \lambda_{TV} L_{TV}(G(\mathbf{x}, \mathbf{z})) \quad (5)$$

$$L_d(\mathbf{r}, \mathbf{s}) = \frac{1}{bs} \sum^{bs} \frac{1}{n^2} \sum^{n^2} (-\log(\text{sigmoid}(\mathbf{s})) \cdot \mathbf{r} - \log(\mathbf{1} - \text{sigmoid}(\mathbf{s})) \cdot (\mathbf{1} - \mathbf{r})) \quad (6)$$

$$\text{sigmoid}(\mathrm{m_{i,j,k}}) = \frac{e^{m_{i,j,k}}}{1 + e^{m_{i,j,k}}} \quad \text{for } i \in (0, 1, \ldots, bs) \text{ and } j, k \in (0, 1, \ldots, n) \quad (7)$$

Here, L_{MAE} is the mean absolute error (L1 error) between artificially generated output $G(x, z)$ and true Hi-C submatrices y (as grayscale images), L_d is the discriminator loss according to eq. (6), in this case with respect to artificially generated matrices, and l_{TV} is a total variation loss as in eq. (1) and eq. (12). Since the discriminator is a patchGAN discriminator, it is working on batches which contain bs image patches of shape (n, n), and $\mathbf{1}$ is also a tensor of shape (n, n), populated with ones. To obtain a scalar loss value, the mean was taken across both batches and patches. The sigmoid function was computed individually for all elements $m_{i,j,k}$ in the (bs, n, n)-shaped tensors as shown in eq. (7), transforming them into value range [0...1]. Finally, individual scalar factors λ were used to balance the influence of the individual loss portions on the joint loss function, cf. section 4.8.1.

As already indicated in fig. 7, the discriminator was trained as proposed by Isola et al., meaning that the binary cross entropy loss was optimized with respect to true- and fake samples, eq. (8) .

$$L_D(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \lambda_D * [L_d(\mathbf{1}, D(\mathbf{x}, \mathbf{y})) + L_d(\mathbf{0}, D(\mathbf{x}, G(\mathbf{x}, \mathbf{z}))] \tag{8}$$

In eq. (8), L_d is again the PatchGAN loss defined above in eq. (6), y are true Hi-C submatrices, $G(x, z)$ are Hi-C submatrices artificially generated by the generator G from chromatin feature data x and random noise z, and $\mathbf{1}, \mathbf{0}$ are tensors of the appropriate shape with all ones and all zeros, respectively. λ_D is a scalar factor proposed by Isola et al. to slow down the learning process of the discriminator compared to the generator.

The original idea behind combining mean absolute error L_{MAE} and PatchGAN adversarial loss L_d is that optimizing MAE mainly minimizes low-frequency errors, while optimizing the PatchGAN-loss with its small patches helps reducing high-frequency errors [49]. The TV loss has been added for edge-aware noise removal, in line with other works in the field [43], cf. section 3.1.3.

Both the generator- and discriminator loss functions were optimized with the Adam optimizer, alternately training discriminator and generator, see section 4.8.1 for details.

Compared to other cGAN architectures, *pix2pix* is rather simple, but well studied for different applications like label-to-image transfer, sketch-to-image transfer, grayscale-to-color transformations or image infill tasks [49]. Since the problem at hand has likely not been tackled by a tailor-made GAN yet, choosing a "general-purpose" cGAN as a starting point seemed reasonable. Another advantage of *pix2pix*, and U-Net architectures in general, is that they can achieve good performance even with comparatively few training samples [49, 66].

3.2.2 Using a DNN for feature embedding

The embedding network required for processing the conditional inputs has on obvious mandatory property: it must be capable of embedding the input of shape $(3w, n)$ into a 2D grayscale image of shape $(w, w, 1)$ which can then be processed by a *pix2pix*-like network. Naturally, this can be achieved by using the dense neural network explored above and reshaping its "upper triangular" output vector back into a symmetric "image"-tensor with the required shape.

The rationale here was to first use the DNN to get a coarse estimate of the predicted matrix, and then the cGAN to refine the results, somewhat similar to the two-stage approach by Zhang et al. [63], albeit much less sophisticated. Naturally, this approach also allows to pre-train the dense network as discussed above in section 3.1, this time for window size $w = 64$ and then use transfer learning to provide better starting values for the weights of the embedding network, potentially improving both stability and convergence speed of the cGAN network.

A disadvantage of the dense network approach is that the number of neurons in its output layer quadratically depends on the window size w, cf. section 3.1.1 and 4.7.1. Taken together with the three fully connected layers underneath, this leads to a superlinear increase in the number of parameters along with w, further aggravated by the fact that the cGAN requires w to be powers of two, cf. section 4.8.1 and 4.8.2 (table 4, p. 36). The DNN embedding has thus only been explored for $w = 64$, both with and without pre-training the DNN. The results are to be found in section 5.2.1 and some more technical details are given in section 4.8.2.

3.2.3 Using a CNN for feature embedding

Since the results from the cGAN with DNN embedding were not convincing, another embedding network was designed, based on convolutional layers. Here, the idea was to have a trainable embedding that keeps the localization information contained in the chromatin feature data, has less parameters than a dense embedding and can efficiently be trained along with the rest of the network even at window sizes up to 256 bins.

Since no example for such an embedding was found in literature, a first attempt was made with a simple convolutional neural network (CNN). This network essentially featured 8 building blocks, each consisting of 1D convolution, batch normalization layer to avoid overfitting and leaky ReLU activation [69] to avoid the so-called "dying ReLU" problem. A final 9th convolution layer was added to ensure the required output shape. The complete setup is shown in section 4.8.3 (fig. 18, p. 45). Compared to the dense approach, the number of trainable parameters in the CNN embedding is widely independent of the window size and stays within 4.24...4.29 million parameters for the investigated window sizes 64, 128 and 256, cf. table 4 (p. 36).

The results of the cGAN with CNN embedding are shown in section 5.2.2 and were indeed better than the ones obtained with DNN embeddings.

3.2.4 Using mixed DNN- and CNN-embedding for feature embedding

Besides using a DNN- or CNN-embedding for *both* generator and discriminator, it is obviously also possible to use different embedding network types. To investigate the influences of the embedding network on the general performance, a "mixed" embedding, i. e. a DNN-embedding for the generator and a CNN-embedding for the discriminator was tested, using the same DNN and CNN as above. The results are shown in section 5.2.3.

4 Methods

In the following sections, the methodical details for the tasks within the thesis will be discussed in more detail. First, input data and data preprocessing will be discussed, followed by an explanation of the quality metrics used to assess the predictions of Hi-C matrices made throughout the thesis. Next, sections 4.7 and 4.8 will give details with regard to the two different approaches made in this thesis to advance the state of the art in predicting Hi-C matrices.

4.1 Input data

For the thesis at hand, data from human cell lines GM12878 and K562 was used. The exact data sources and data preprocessing for the Hi-C matrices and ChIP-seq data will be outlined below

Hi-C data due to Rao et al. [11] was downloaded in .hic format from Gene Expression Omnibus under accession key GSE63525. Here, the quality-filtered "combined_30" matrices were taken, which contain only high-quality reads from both replicates.

Next, matrices at 5 kbp binsize were extracted and converted to cooler format using `hic2cool` and subsequently coarsened to resolutions of 10, 25, 50 and 100 kbp using `cooler coarsen`, see listing 1. Contrary to the work from Farré et al. [30], which is using ICE plus distance-based normalization, and many others in the field which are using ICE- or KR-normalization, these matrices have not been normalized for the thesis at hand because no benefit of doing so was found during the study project [13].

ChIP-seq data for 13 chromatin features and DNaseI-seq data was used, cf. table 2. Here, the data was downloaded in .bam format either via the ENCODE project [16, 17] or directly from the file server of the University of California (UCSC). In all cases, bam-files for replicate 1 and 2 were downloaded in their most recent versions (if applicable); the UCSC- (wgEncode...) or GEO- (GSM...) identifiers are also given in table 2. For convenience, the pdf version of this document also provides download links for concrete files in section 7.1.

After downloading, the bam files were converted to bigwig format, which was found more convenient to handle, and the replicates were merged into one bigwig file by taking the mean, as in the study project [13]. Pseudocode for the full conversion process is given in listing 2.

The choice of chromatin features is widely in line with the work by Zhang et al. [28]; aside structural proteins like CTCF and Cohesin subcomponents RAD21 and SMC3, active/passive markers are used as well.

4.2 Sample generation process for the dense neural network

This thesis follows the sliding window approach proposed by Farré et al. [30].

First, all chromatin features were binned to binsize b_{feat} by splitting each chromosome of size cs into $l_{feat} = \left\lceil \frac{cs}{b_{feat}} \right\rceil$ non-overlapping bins of size b_{feat} and taking the mean feature value within the genomic region belonging to each bin. All n chromatin factors were processed in this way and

feature name	cell line	
	GM12878	K562
CTCF	GSM733752	GSM733719
DNaseI	wgEncodeEH000534	wgEncodeEH000530
H3k27ac	GSM733771	GSM733656
H3k27me3	GSM733758	GSM733658
H3k36me3	GSM733679	GSM733714
H3k4me1	GSM733772	GSM733692
H3k4me2	GSM733769	GSM733651
H3k4me3	GSM733708	GSM733680
H3k79me2	GSM733736	GSM733653
H3k9ac	GSM733677	GSM733778
H3k9me3	GSM733664	GSM733776
H4k20me1	GSM733642	GSM733675
Rad21	wgEncodeEH000749	wgEncodeEH000649
Smc3	wgEncodeEH001833	wgEncodeEH001845

Table 2: Chromatin features used for the thesis

then stacked into a (l_{feat}, n)-shaped array. Here, the number of chromatin features was constant for all investigations within this thesis, $n = 14$ (cf. section 4.1).

Separate Hi-C matrices for each chromosome were derived from the cooler format as (l_{mat}, l_{mat})-shaped matrices, $l_{mat} = \left\lceil \frac{cs}{b_{mat}} \right\rceil$ being the number of bins in the given chromosome. Initially, $b_{feat} = b_{mat}$ was used, which leads to $l_{feat} = l_{mat}$, because cs is a constant for a given chromosome.

A sliding window approach was then applied to generate training samples for the neural networks G described below. Here, subarrays of size $(3w_{mat}, n)$ were cut out from the feature array such that the i-th training sample corresponded to the subarray containing the columns i, $i + 1$, $i + 2$, ..., $i + 3w_{mat}$ of the full array. Sliding the window along the array with stepsize one obviously yields $N = l - 3w_{mat} + 1$ training samples. The corresponding Hi-C matrices were then cut out along the diagonal of the original matrix as submatrices with corner indices $[j,\ j]$, $[j,\ j + w_{mat}]$, $[j + w_{mat},\ j + w_{mat}]$, $[j + w_{mat},\ j]$ in clockwise order, where $j = i + w_{mat}$. The idea here is that the first 0, 1, ... w_{mat} columns of each feature sample form the left flanking region of the training matrix, the next $w_{mat} + 1$, $w_{mat} + 2$, ... $2w_{mat}$ correspond to the matrix' region and the last $2w_{mat} + 1$, $2w_{mat} + 2$, ... $3w_{mat}$ columns form the right flanking region. Since Hi-C matrices are symmetric by definition, only the upper triangular part of the submatrices was used, flattened into a $w \cdot (w + 1)/2$ vector.

Figure 8 exemplarily shows the sample generation process for a matrix of shape (16,16) with $w_{mat} = 4$ and $n = 3$ chromatin features. In this case, five training samples would be generated – the one encircled in green and four more to the right, as the window is sliding from left to right until the right flanking region hits the feature array boundary.

The sample generation process for predicting (unknown) matrices was the same as for training, except that no matrix window was generated. Due to the sliding window approach, the output of the network is a set of overlapping submatrices along the main diagonal of the actual target

Hi-C matrix. To generate the final submatrix, all submatrices were added up in a position-aware fashion and finally all values were divided by the number of overlaps for their respective position. Figure 9 exemplarily shows the prediction process for $N = 5$ samples with windowsize $w = 4$ for a matrix of shape (16,16). Note that the first and last w bins in each row (matrix diagonal) always remain empty due to the flanking regions, as do all bins outside the main diagonal and the first $w - 1$ side diagonals. To improve visualisation, all predicted matrices were scaled to value range 0...1000 after re-assembly and stored in cooler format for further processing. Conveniently, cooler also supports storing only the upper triangular part of symmetric matrices, minimizing conversion effort for the data at hand.

Generally, training samples were drawn from chromosomes 1, 2, 4, 7, 9, 11, 13, 14, 16, 17, 18, 20 and 22 of cell line GM12878, validation samples from chromosomes 6, 8, 12 and 15 of cell line GM12878 and test samples from chromosomes 3, 5, 10, 19 and 21 of cell line K562. This approximately implements a 60:20:20 train:validation:test split, see table 3 for details. Reversing training/validation and test cell line, which differ strongly in sequencing depth within the Hi-C experiments, was tested briefly for the cGAN approach and seemed to work, see section 7.4.3. For comparability reasons, two cGANs were additionally trained on single chromosomes 14 and 17 of cell line GM12878, cf. section 4.9. Also for comparability reasons, another cGAN was trained and tested on data from *drosophila melanogaster* embryos, again refer to section 4.9 for details.

Note that windowsize $w = 80$ is likely not suitable for resolutions beyond 50 kbp, because the number of training samples becomes too small to train a network with more than seven million parameters, cf. section 4.7.1.

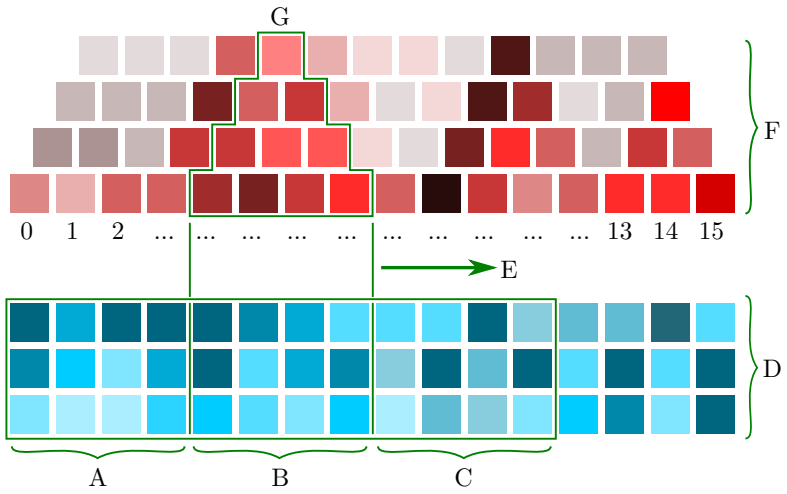
4.3 Sample generation for the cGAN

The samples for the cGAN were generated the same way as the samples for the dense neural network described in section 4.2, with two exceptions. First, Hi-C (sub-)matrices were not entered as vectors corresponding to their upper triangular part, but were instead taken as $(w, w, 1)$-shaped grayscale images with value range [0...1] in 32-bit floating point format. Second, for the cGAN approach the input matrices need not only be square, but their sizes also needed to be powers of two. This ensured the required shapes for the connected up- and downsampling operations in the generator, which essentially are 2D-convolutions and transposed 2D-convolutions with strides two. Within the thesis, windowsizes of $w = \{64, 128, 256\}$ were used, see section 5.2.

4.4 Generalization of feature binning

Most of the binning- and sample generation procedures described above also work for binsize relations $k = \frac{b_{mat}}{b_{feat}} \in \mathbb{N}^{>1}$.

The training matrices remain unchanged, i. e. (l, l)-shaped arrays, from which training submatrices of size (w_{mat}, w_{mat}) can be extracted. With $k \in \mathbb{N}^{>1}$, one bin on the matrix diagonal corresponds to k bins of the feature array, so the feature windowsize needs to be k times the submatrix windowsize, $w_{feat} = k \cdot w_{mat}$. Since the first layer of all neural networks used in this thesis is a 1D convolution, this can be achieved by setting the filter width and strides parameters of the (first) convolutional layer to k, leaving the rest of the network unchanged. However, the



A: left flanking region

B: matrix region

C: right flanking region

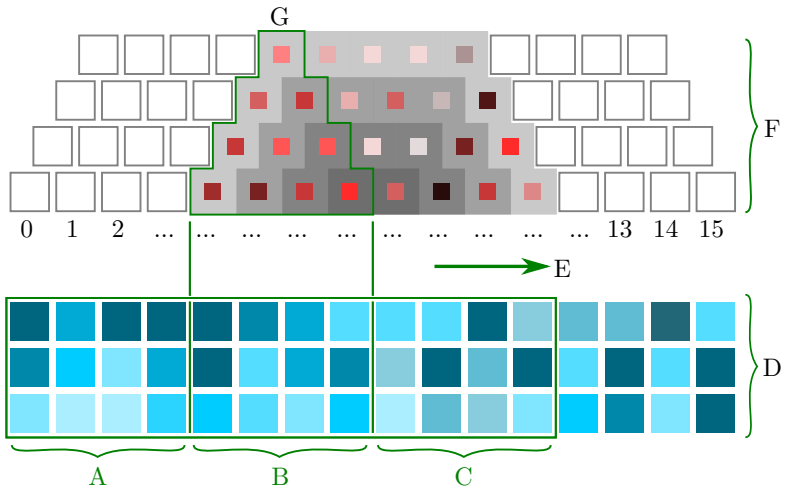
D: chromatin feature array

E: sliding direction for sample generation windows

F: first 4 diagonals of a $(16, 16)$ Hi-C matrix, rotated 45 deg counterclockwise

G: first $(4, 4)$ training matrix, corrsponding to feature window A-B-C

Figure 8: Sample generation process



A: left flanking region

B: matrix region

C: right flanking region

D: chromatin feature array

E: sliding direction for predicted submatrices

F: first 4 diagonals of a predicted $(16, 16)$ Hi-C matrix, rotated 45 deg counterclockwise

G: first predicted $(4, 4)$ matrix

gray backround colors symbolize number of overlapping predictions at that position (bright = 1, dark = 4)

Figure 9: Prediction process

chrom.	length/bp	samples at w = 80 and binsize...					
		5k	10k	25k	50k	250k	500k
1	249250621	49612	24687	9732	4747	759	260
2	243199373	48401	24081	9489	4625	734	248
4	191154276	37992	18877	7408	3585	526	144
7	159138663	31589	15675	6127	2944	398	80
9	141213431	28004	13883	5410	2586	326	44
11	135006516	26763	13262	5162	2462	302	32
13	115169878	22795	11278	4368	2065	222	
14	107349540	21231	10496	4055	1908	191	
16	90354753	17832	8797	3376	1569	123	
17	81195210	16001	7881	3009	1385	86	
18	78077248	15377	7569	2885	1323	74	
20	63025520	12367	6064	2283	1022	14	
22	51304566	10022	4892	1814	788		
∑ train samples		**337986**	**167442**	**65118**	**31009**	**3755**	**808**
6	171115067	33985	16873	6606	3184	446	104
8	146364022	29034	14398	5616	2689	347	54
12	133851895	26532	13147	5116	2439	297	29
15	102531392	20268	10015	3863	1812	172	
∑ valid. samples		**109819**	**54433**	**21201**	**10124**	**1262**	**187**
3	198022430	39366	19564	7682	3722	554	158
5	180915260	35945	17853	6998	3380	485	123
10	135534747	26868	13315	5183	2472	304	33
19	59128983	11587	5674	2127	944		
21	48129895	9387	4574	1687	724		
∑ test samples		**123153**	**60980**	**23677**	**11242**	**1343**	**314**
∑ total samples		**570958**	**282855**	**109996**	**52375**	**6360**	**1309**

Table 3: Training, validation and test samples for sliding window approach

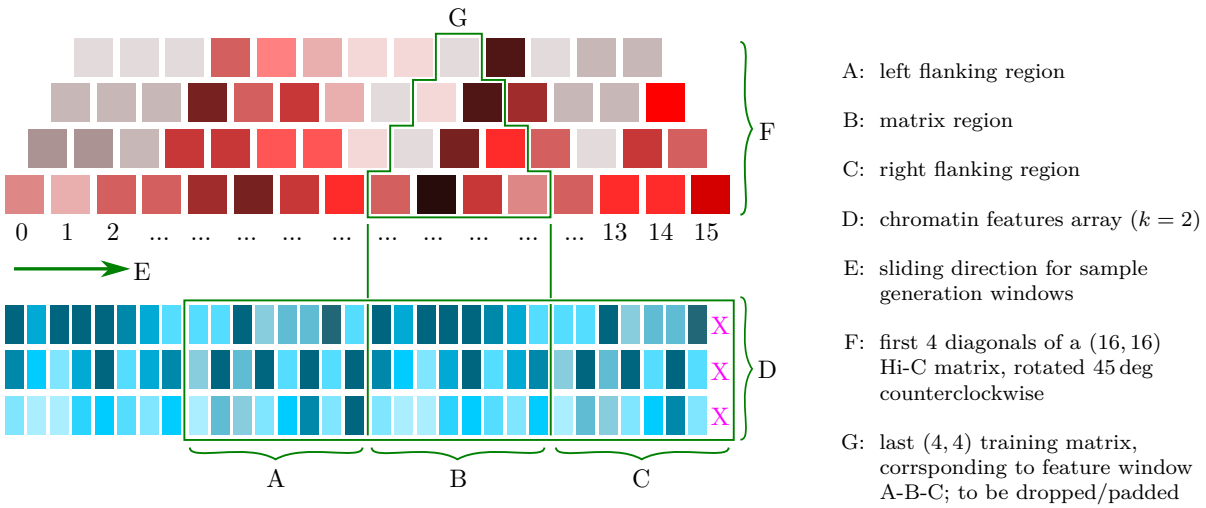


Figure 10: Generalized sample generation process f. $k = 2$

number of bins along the matrix diagonal is generally not k times the number of bins in the feature array, see eq. (9).

$$l_{feat} = \left\lceil \frac{cs}{b_{feat}} \right\rceil = \left\lceil \frac{cs}{k \cdot b_{mat}} \right\rceil \neq k \cdot \left\lceil \frac{cs}{b_{mat}} \right\rceil = k \cdot l_{mat} \tag{9}$$

For the training process, this discrepancy was resolved by simply dropping the last training sample, if the feature window belonging to it had missing bins. For the prediction process, the feature array was padded with zeros on its end. This procedure ensures that no errors are introduced into the *training* process by imputing values, but keeps the size of the *predicted* matrix consistent with the training matrix binsizes.

Figure 10 shows the generalized training process with a $(16, 16)$-training matrix and $k = 2$. If $15 \cdot b_{mat} + 1 \leq cs < 15 \cdot b_{mat} + b_{feat}$ held for the chromosome size cs in the example, then the number of bins on the matrix diagonal would be $l_{mat} = 16$, while the number of chromatin feature bins would be $l_{feat} = 31 \neq 2 \cdot l_{mat}$. In this case, the 5th sample would be dropped for training, while a column with zero bins – symbolized by pink crosses in fig. 10 – would be added to the feature array for prediction so that the resulting matrix would still have the desired size of $(16, 16)$.

4.5 Quality metrics for predicted Hi-C matrices

Assessing the quality of synthetically generated images is a long-standing problem, which has not yet been solved, see e g. [61, p. 19]. Within this thesis, distance-stratified Pearson correlations were computed between predicted and real matrices to measure the quality of the predictions. While the suitability of suchlike correlations as a quality metric for Hi-C matrices seems debatable in general [70], distance stratified Pearson correlation is quite common in the field of Hi-C matrices and thus useful at least for comparisons with other approaches.

To compute the correlations, the w bins at the left and right boundaries of the investigated matrices were first removed, because these never contain valid values due to the chosen sample

generation approach, cf. section 4.2, fig. 9 and 4.3. Next, the remaining values were grouped into vectors according to their genomic distance d, up to the maximum distance w, and then Pearson correlations ρ were computed separately for each vector as usual, eq. (10). Here, $true|_d$ and $pred|_d$ are the true and predicted values restricted to bin distance d, while $\sigma(true|_d)$ and $\sigma(pred|_d)$ are the respective standard deviations and $cov(\cdot, \cdot)$ is the covariance.

$$\rho(true|_d,\ pred|_d) = \frac{cov(true|_d, pred|_d)}{\sigma(true|_d)\ \sigma(pred|_d)} \tag{10}$$

Since a numerically stable and efficient implementation of the Pearson correlation is non-trivial, the corresponding function from the pandas library XXXwas used for actual calculations. An obvious disadvantage of distance-stratified Pearson correlations is that they are undefined if either the predicted matrices or – not observed in practice – the true matrices have the same value, and thus zero standard deviation, for a certain distance.

In addition to the correlations themselves, the area under the correlation curve (AUC) was computed for all curves in each plot to obtain a single-valued, albeit very abstract quality metric. In line with Zhang et al. [28], trapez integration method was used for numerical area computations and the values were normalized to the interval [-1, 1] by dividing through the maximum distance to allow for comparisons across experiments. Furthermore, all Pearson correlation plots show the correlation between the "true" GM12878 matrix and the corresponding target matrix as a baseline, i. e. the Pearson correlation that is obtained when simply using the corresponding GM12878 matrix and chromosome as a "prediction" for a given target matrix and chromosome.

In computer graphics, other similarity metrics like the structural similarity index (SSIM), peak signal-to-noise ratio (PSNR) or, especially for images generated by GANs, Fréchet inception distance (FID) seem more common than Pearson correlation. However, their suitability for plots of Hi-C matrices seems not to have been investigated so far. Unlike Pearson correlation, SSIM and PSNR both depend on absolute values in the matrices to be compared, so that for example $PSNR(\mathbf{X}) \neq PSNR(k \cdot \mathbf{X})$ for $k \in \mathbb{R}^{>1}$ and X a matrix of appropriate shape, which seems not optimal for a use case where structures are more interesting than actual values. Computing FID would require reshaping outputs into images of shape (299, 299, 3) to meet Inception v3 specifications [71]. This is a non-trivial task for the given matrix plots, where the longer edge – defined by $chromsize/binsize$ – is usually much longer and the shorter edge – defined by the windowsize w – is usually shorter than 299 bins, or pixels, respectively. While computing the FID for individual (w, w)-submatrices would be possible, generalizing such results to the whole matrix seems without example in literature so far.

Since relying on one metric alone seemed problematic, selected areas of the test set were plotted against the true Hi-C matrices in these areas, see section 4.6 below. Note that visual comparison of predicted Hi-C matrices and true Hi-C matrices in this setting is strongly influenced by the value range of the matrices, while the Pearson correlation is generally not.

4.6 Matrix plots

To allow for a visual comparison, selected areas of the predicted matrices were plotted against their true counterparts using pygenometracks [72]. Since it was impossible to plot the full test

set every time due to the amount of figures needed, three exemplary regions were selected, chromosome 21, 19 and 5, 30 to 40 Mbp, respectively. These regions were chosen because they feature both small and large TADs as well as nested TADs and subregions with very little chromatin feature signal values.

In general, most matrix plots additionally feature the sign of the first eigenvector in a purple track, named "binarized PCA" (values +1 or -1) and the summarized value of all 14 chromatin features in a green track, named "feature sum", see e.g. fig. 22. Feature sum was computed by binning all chromatin features at 25 kbp and summing up the values across all features for each bin. Eigenvectors were computed using `hicPCA` [60] and then the sign for each 25 kbp bin was extracted using a simple python script. Finally, eigenvectors and feature sum were written out in bedgraph format, which pygenometracks can plot natively.

All tracks were plotted with an original resolution of 200 dpi and a diagram width of 150 mm. For the interaction counts in the Hi-C matrix plots, "log1p" transformation was applied and the true matrices were inverted, i. e. reflected at the diagonal. No minimum or maximum values were set for plotting, but note that predicted matrices were scaled to value range 0...1000 before they were stored, cf. section 4.2.

4.7 Dense neural network approach

In the following sections, the setup for the Dense Neural network inspired by Farré et al. [30] and its variations will be discussed in detail.

4.7.1 Basic setup

The basic setup for the dense neural network approach closely follows the proposal by Farré et al. [30], so a windowsize of $w = 80$ and $b_{feat} = b_{mat} = 10$ kbp was initially used. With these parameters, the network has a total of 7 486 436 trainable weights in five trainable layers; one convolutional layer and four densely connected layers, fig. 11. For brevity, the sigmoid activation after the 1D convolution and the ReLU activations after the Dense layers are not shown. The dense layers all use a "L2" kernel regularizer and all dropout layers have a dropout probability of 10%.

The training goal for the neural network G is to find weights ω^* for its five trainable layers such that the mean squared error L_2 between the predicted submatrices $\mathbf{M}_s = G_s(\omega)$ and the training submatrices $\mathbf{T}_s$ becomes minimal for all N training samples $s \in (1, 2, \ldots, N)$. Here, $\mathbf{M}_s$ is given by the activations of the last dense layer, which are to be interpreted as the upper triangular part of a symmetric (w, w) Hi-C matrix. Formally, one needs to optimize

$$\omega^* = arg\,min_\omega L2(\omega) = arg\,min_\omega \frac{1}{N} \sum_{s=0}^{N} (\mathbf{M}_s - \mathbf{T}_s)^2 \tag{11}$$

For the thesis at hand, stochastic gradient descent (SGD) with learning rate 10^{-5} was used to find ω^*. Initial values ω_{init} were drawn from a Xavier initializer, a uniform distribution with limits depending on the in- and output shapes. Following [30], optimization was performed on minibatches of 30 samples, assembled randomly from the N training samples to prevent

location-dependent effects and improve generalization. For training, the last batch was dropped, if $N/30 \notin \mathbb{N}$.

The network and its learning algorithm were implemented in python using tensorflow deep learning framework, partially with keras frontend [73, 74], see repository XXX. All computations were performed on virtual machines with different properties, partially with GPU assistance; see 7.3 for details.

4.7.2 Modifying kernel size, number of filter layers and filters

For the "wider" variant, the kernel size of the of the 1D convolutional layer was increased to $4k$ with strides k, where $k = b_{mat}/b_{feat}$ is the relation beetween matrix- and feature binsize. Mirror-type padding was used to maintain the output dimensions of the basic network, which had 7 486 478 trainable weights for $k = 1$.

For the "longer" variant three 1D-convolutional layers with 4, 8 and 16 filters were used in place of the basic network's single convolutional layer, cf. fig. 12. This replacement was also made for the "wider-longer"-variant, additionally increasing the respective kernel sizes to $4k$ (with strides k), 4 and 4, cf. fig. 12 (right). In both cases, the dropout rate was increased to 20%. The "longer" variant had 9 142 665 trainable weights and the "wider-longer" had 9 143 313 for $k = 1$.

For the variant with generalized binning, features were binned at $b_{\text{feat}} = 5\,\text{kbp}$ while keeping the matrix binsize $b_{\text{mat}} = 25\,\text{kbp}$, so the factor for the relation between the two binsizes was $k = 25/5 = 5$. This yields input feature arrays of size $(3w \cdot k, n = (3 \cdot 80 \cdot 5, 14) = (1200, 14)$. Replacing the first convolutional layer of the basic network by a 1D convolutional filter with kernel size $k = 5$ and strides $k = 5$ without padding, this input was again compressed to a $(3w, 1) = (240, 1)$ tensor and fed into the flatten layer, cf fig. 11. The rest of the network remained the same so that the number of trainable parameters increased only slightly to 7 486 492.

4.7.3 Combination of mean squared error, perception loss and TV loss

For computing the combined MSE, perception- and TV-loss, input features were first passed through the network as normal, and mean squared error was computed on the predicted "upper triangular vectors" vs. the real vectors. Next, both the output- and target vectors were converted to symmetric grayscale images by embedding them into $(w, w, 1)$-shaped tensors, where w is again the windowsize.

For a pixel-image $\mathbf{y}$, total variation can be defined as the sum of the absolute differences for neighboring pixel-values, eq. (12) [55]

$$tv(\mathbf{y}) = \sum_{i,j} \sqrt{|y_{i+1,j} - y_{i,j}|^2 + |y_{i,j+1} - y_{i,j}|^2} \tag{12}$$

For efficiency, the tensorflow implementation from tf.image.total_variation was used, taking the sum across batches as loss value, as recommended in the tensorflow documentation [75].

For perception loss, the predicted images and the true images were first fed through a pre-trained *VGG-16* network with fixed weights, truncated after the third convolution layer in the

fourth block ("block4_conv3"), the last layer used by the influential work of Johnson et al. [54]. The loss was then computed as mean squared error between the "predicted" and "true" output activations of the truncated VGG-16 network.

Let $\mathbf{M}_s = G_s(\boldsymbol{\omega})$ again be the output of the neural network G described above, and $\mathbf{T}_s$ the true matrices for training samples s in vector form, and let $\mathbf{M}'_s$ and $\mathbf{T}'_s$ be their grayscale image counterparts as described above. Furthermore, let $tv(\mathbf{x})$ be the total variation of image $\mathbf{x}$ and $vgg(\mathbf{x})$ the output of the perception loss network for image $\mathbf{x}$. The optimization problem for the modified network was then formulated by means of eq. (13) to find weights $\boldsymbol{\omega}^*$ such that

$$\boldsymbol{\omega}^* = arg\,min_\omega(\lambda_{MSE}\frac{1}{N}\sum_{s=0}^{N}(\mathbf{M}_s - \mathbf{T}_s)^2 + \lambda_{TV}\sum_{s=0}^{N} tv(\mathbf{M}'_s) + \lambda_{VGG}\frac{1}{N}\sum_{s=0}^{N}(vgg(\mathbf{M}'_s) - vgg(\mathbf{T}'_s))^2) \tag{13}$$

Weight initialization for network G and minibatching was done as described in section 4.7.1. The *VGG16* network and the corresponding weights were taken from the keras implementation pre-trained on *ImageNet* XXX. As a side effect, the usage of *VGG16* imposes the restriction $w \geq 32$ on the windowsize w, which is not problematic, since again $w = 80$ was chosen for all experiments.

The network $G(\boldsymbol{\omega})$ could in principle be any of the variants described above in section 4.7.2, but for the thesis at hand, only the initial network from section 4.7.1 was used.

4.7.4 Combination of mean squared error and TAD-score-based loss

To optimize both for mean-squared error and a TAD-score-derived loss, the following optimization task was defined to find weights $\boldsymbol{\omega}^*$ such that

$$\boldsymbol{\omega}^* = arg\,min_\omega(\lambda_{MSE}\frac{1}{N}\sum_{s=0}^{N}(\mathbf{M}_s - \mathbf{T}_s)^2 + \lambda_{score}\frac{1}{N}\sum_{s=0}^{N}(score(\mathbf{M}'_s, ds) - score(\mathbf{T}'_s, ds))^2 \tag{14}$$

where $\mathbf{M}_s$ is again the Hi-C submatrix predicted by the network $G(\omega)$ for sample s and $\mathbf{T}_s$ is the corresponding true Hi-C submatrix.

To compute the TAD-score-based loss $score(\cdot,\cdot)$, the predictions and true Hi-C matrices $\mathbf{M}$ and $\mathbf{T}$ in vector form (upper triangular part) were first converted back to complete, symmetric Hi-C matrices $\mathbf{M}'$, $\mathbf{T}'$. Next, in a custom network layer, all diamonds with size ds inside the submatrices of size w were cut out using tensor slicing and the values inside the diamonds were reduced to their respective mean. This yields score vectors – more exactly, tensors with shape $(w - 2\,ds, 1)$. After computing the latter for both predicted- and real Hi-C submatrices, the mean squared error between them was computed as usual and weighted with a user-selected loss weight λ_{score}, see eq. (14).

While it would also have been possible, and probably faster, to slice the outputs of the networks directly in vector form, this is rather unintuitive and was therefore not implemented for the thesis.

4.8 Hi-cGAN approach

In the following subsection, the setup and training process of the cGAN network developed for this thesis and the respective embeddings for the conditional input – the chromatin features – will be described in more detail. First, the basic network setup based on the well-known *pix2pix* cGAN architecture [49] and the training process are explained in section 4.8.1, then the respective embedding networks are discussed in subsections 4.8.2, 4.8.3 and 4.8.4.

4.8.1 Modified pix2pix network

Several implementations of the original *pix2pix* network [49] are publicly available, usually for the original image size of 256×256. For the thesis at hand, implementation concepts from two tutorials [76, 77] were combined and the code was adapted to the given requirements.

Within the generator, two of the down- and upsampling layers inside the U-Net portion were made optional to allow processing smaller images of sizes 64×64 and 128×128, see fig. 13. Note that the generator (still) only supports square images with edge lengths that are powers of two and at least $2^6 = 64$. Furthermore, symmetry of the output images was enforced by adding their transpose and multiplying by 0.5, cf. [35]. The down- and upsampling layers shown in fig. 13 are custom blocks detailed in fig. 15 and 16. All 2D-convolutions and 2D-deconvolutions had kernel size $(4, 4)$ and were initialized with values drawn from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.02$. Finally, the output layer of the generator was changed from tanh- to sigmoid activation. This empirically showed better results, probably because the training- and test matrices were scaled to 0...1, which is also the value range of the sigmoid function.

Compared to the original *pix2pix* setup, one downsampling layer was omitted in the discriminator for windowsize $w \in \{128, 256, \dots\}$ and another one for $w = 64$. This made the discriminator patches larger, especially for the smaller windowsizes, cf. fig. 14, and was empirically found to improve the results slightly in the given application. Symmetry was enforced after all convolutions in the same way as in the generator. Kernel sizes and initializations for all 2D convolutions also were the same as for the generator, and leaky ReLU-activations were used with parameter $\alpha = 0.2$, as in all downsampling layers.

Both the discriminator and the generator featured their own, trainable embedding network to convert the conditional input, i. e. the chromatin feature data of shape $(3w, n)$, into grayscale images of shape $(w, w, 1)$. These networks will be discussed below in section 4.8.2 and section 4.8.3.

The loss functions for the generator and discriminator were implemented as shown in equations (5) to (8) with parameters $\lambda_{adv} = 1.0, \lambda_{MAE} = 100.0, \lambda_{TV} = 10^{-12}$ and $\lambda_D = 0.5$. Optimization was performed on minibatches of size 32, 8 and 2 for windowsizes 64, 128 and 256, respectively, using Adam optimizers with learning rate $2 \cdot 10^{-5}$ for the generator, 10^{-6} for the discriminator and $\beta_1 = 0.5$ for both generator and discriminator. The choice of batchsizes was partially dictated by memory limitations of the available GPUs, cf. section 7.3.

a) the number of trainable parameters XXX

4.8.2 Using a DNN for feature embedding

In order to use the DNN described in section 4.7.1 as an embedding network for the cGAN, only small amendments were required to adjust the input shapes, i. e. to provide symmetric Hi-C matrices as grayscale images instead of the upper-triangular-vector representation native to the DNN, see fig. 17. The triu-reshape layer is a custom tensorflow network layer which generates an output tensor of appropriate shape (w, w) and sets its upper triangular part to the values given by the input vector. Symmetrization was then performed by adding this tensor to its transpose and dividing the values on the diagonal by two, because the diagonal is contained both in the upper triangular part of the matrix and its transpose. Finally, the required third axis was added to get the shape of a grayscale image. The number of trainable parameters for the DNN embedding is shown in table 4. Note that all trainable parameters stem from the DNN here; the reshaping layers do not have any trainable parameters.

windowsize	**trainable weights**	
	CNN	DNN
64	4243968	5502796
128	4260416	16034732
256	4293312	57877612

Table 4: Trainable weights for embedding networks

For the thesis, the DNN-embedding was used in two ways. First, it was trained together with the rest of the cGAN with weight initialization as described in section 4.7.1 and 4.8.1. Second, a DNN was pre-trained as described in section 4.7.1 with windowsize $w = 64$ and the learned weights were transferred to the cGAN once before the start of the training process, which was then continued as described in section 4.8.1. The results of the pre-training are visualized in section 7.4.2 (fig. 61); the state after 250 epochs was the one transferred to the cGAN.

DNN embeddings were only used with windowsize $w = 64$ due to the large number of parameters to be trained at windowsizes 128, 256 and higher.

4.8.3 Using a CNN for feature embedding

The convolutional embedding network consists of 8 convolutional blocks and a final 1D convolution layer, as shown in fig. 18. Each of the convolution blocks start with a 1D convolution with kernel size 4, strides 1, padding “same” and “L2” kernel regularization with parameter $l = 0.02$, followed by batch normalization and leaky ReLU activation with parameter $\alpha = 0.2$. The last 1D convolution consists of w filters with kernel size 4, strides 3 and padding “same”, followed by sigmoid activation; this last convolution layer was not using kernel regularization. All kernel weights of the 1D convolutions in the embedding network were initialized by a Xavier initializer.

In the CNN-embedding, the number of trainable parameters is constant for all layers, except for the last convolutional layer, where the number of convolutional filters is equal to the windowsize, cf. fig. 18. For the three windowsizes $w = \{64, 128, 256\}$ used within this thesis, the CNN-embedding network contained about 4.2 to 4.3 million trainable weights, see table 4 for details.

4.8.4 Mixed embedding for generator and discriminator

In the mixed setting, the dense neural network described above in section 4.8.2 was used as embedding network for the generator and the CNN described in section 4.8.3 was used as embedding network for the discriminator. As with the DNN alone, the mixed setting was used both with and without weight transfer. In the latter case, again the weights of the DNN were replaced by the ones of the same pre-trained DNN as in section 4.8.2 before the training process started.

4.9 Comparsion with other approaches

To compare our results with the ones due to Zhang et al. [28], the available predictions for cell line K562, chromosomes 14 and 17 were downloaded from Zenodo XXX. Here, results from the "WINDOW" and "MULTICELL" approaches were selected, whereby the "WINDOW" method was using training data from GM12878, chromosome 14 or 17, while the MULTICELL approach was using chromatin feature data from cell lines GM12878, HMEC, HUVEC, K562(!) and NHEK with training matrices from GM12878, chromosome 14 or 17. These results were deemed to offer the best comparability to the cGAN- and DNN approaches of this thesis. The downloaded data is in text format and was first converted into cooler format, whereby a surprising sparsity of the data was noted. For chromosome 14, only 42.4 % and for chromosome 17, only 59.9 % of all possible interacting pairs for the chosen windowsize $w = 200$ at binsize $b = 5000$ were contained in the datasets, and no interacting pairs with predicted interaction counts below 0.01 were found in the data. Instead, the value range was between 0.36 and 5.10 across all four datasets, with similar values for each dataset alone. The missing pairs were found to be distributed all across the chromosome and all distance ranges, exept for distance zero, i. e. the Hi-C matrix diagonal, which was not contained in the datasets at all. For the conversion to cooler format, the missing values were assumed to be zero. Next, the cooler matrices were coarsened to binsize 25 kbp to allow comparisons with the cGAN- and DNN results using cooler coarsen.

For the comparison between *HiC-Reg* and Hi-cGAN, two additional cGAN models with CNN embedding were trained on chromosome 14 and 17, respectively, using (arbitariliy selected) chromosome 10 for validation. Due to the small number of training samples for in this setting, windowsize was set to $w = 64$ and batch size to $bs = 2$. The rest of the training process remained the same, cf. section 4.8.1. Additionally, HiC-Reg was compared to a cGAN-CNN model with windowsize $w = 256$ and CNN embedding, trained on the typical training data set, which includes chromosomes 14 and 17 besides others, cf. sections 4.8.1, 4.2 and 4.3.

When computing the distance-stratified Pearson correlations only from the available (non-zero) predicted values in the dataset XXX, the results for chromosome 17 showed very good accordance with the data published in the paper [28, fig. 10], fig. 19, allowing the conclusion that the correlation computations should in principle be comparable between this thesis and the paper.

For reference, training and prediction were re-done for the WINDOW approach and cell lines GM12878 $\rightarrow$ K562, chromosome 17. With regard to chromatin features, the same bam files were taken as in section 4.1, except for SMC3, which was replaced by TBP, in line with the paper by Zhang et al. [28]. The bam files were then converted to text files using custom bash scripts and

the `aggregateSignal` application provided by HiC-Reg, cf. listing 3 and 4. For simplicity, only replicate 1 was used in all cases. With regard to matrix data, cooler files were first prepared as described in section 4.1. Next, square root vanilla coverage correction was applied, the matrices were dumped into text files and a custom python script was used to convert the text inputs to meet HiC-Reg's requirements, cf. listing 5 and 6. For the rest of the training- and prediction process, the instructions in HiC-Reg's github repository were followed. The resulting text files were converted back to cooler using another custom python script, see listing 7. The sparsity in the results was 59.6 %, confirming that filtering occurs during the sample generation- or training process of HiC-Reg, although it is not explicitly mentioned in the paper. In terms of Pearson correlations – here, including zero-values – the recomputed results from replicate 1 were slightly better, but overall very similar to the ones restored from the published dataset XXX.

It seems strange that zero-values have not been considered when computing the correlations for the publication [28]. Correctly predicted zero-values strongly increase the quality of the output matrices and should thus be included in the correlation computations. Pearson correlations for the predictions in section 5.3 were thus computed for full chromosomes, including zero-values.

To compare our results with the ones from Farré et al. [30], the corresponding Hi-C- and chromatin feature data for drosophila melanogaster were downloaded and pre-processed as described in the following paragraphs.

Hi-C data from 16-18 hours old embryos was downloaded in text format from Schuettengruber et al. [48] from the gene expression omnibus, accession XXXfor binsize 10 kbp and subsequently converted to cooler format using a custom python script, XXX.

With respect to *chromatin features*, raw reads from modEncode ChIP-seq experiments with 14-16 hour old embryos [78] were obtained from sequence read archive (SRA) for 49 features in fastqsanger format, using the accession keys given in table 5. Note that while Farré et al. state they have used 50 features, in fact only 49 unique features are specified in their paper, H3k4me1 being listed twice [30, p. 9], which explains why table 5 has 49 instead of the expected 50 entries. After downloading, the reads were mapped to the drosophila melanogaster reference genome dm3 using bowtie2 mostly with default parameters, see listing 8 for details, and the resulting bam files were converted to bigwig format as described in section 4.1 and listing 2. While most experiments feature 2 to 4 replicates, only replicate 2 was considered for each chromatin feature to limit disk space usage and processing time.

The bigwig files were then used as inputs for the DNN and cGAN as described above in sections 4.2, 4.3, 4.7 and 4.8. Here, the DNN was used in its initial configuration at windowsize $w = 80$ and the cGAN was used with CNN embedding at windowsize $w = 64$. Both the DNN and cGAN were trained on samples from chromosome 2L and 2R, samples from chromosome 3L were used for validation and predictions were made on chromosomes 3R and X using a batchsize of 2 for the cGAN and 30 for the DNN. All other parameters were set to the same values as in sections 4.7 and 4.8. Note that the windowsizes specified above are the maximum reasonable values, since the Hi-C data due to Schuettengruber et al. [48] is limited to distances below approximately 80 bins for unknown reasons, and the cGAN approach is in turn restricted to power-of-two windowsizes.

Unfortunately, Farré et al have not published raw data from which (predicted) Hi-C matrices and Pearson correlations could have been reconstructed. For this reason, the respective matrix plots and Pearson correlation graphs were extracted from the publication and added to the

figures generated by the DNN- and cGAN approaches by means of image processing. Note that the correlations are in this case given *per position* and cannot be compared to the distance-stratified correlations used everywhere else in this thesis. However, the usual distance-stratified correlations for all test chromosomes are shown in the appendix, figures XXXand XXX. Like the distance-stratified-correlation plots, per-position-correlation figures were generated by a custom python script XXX.

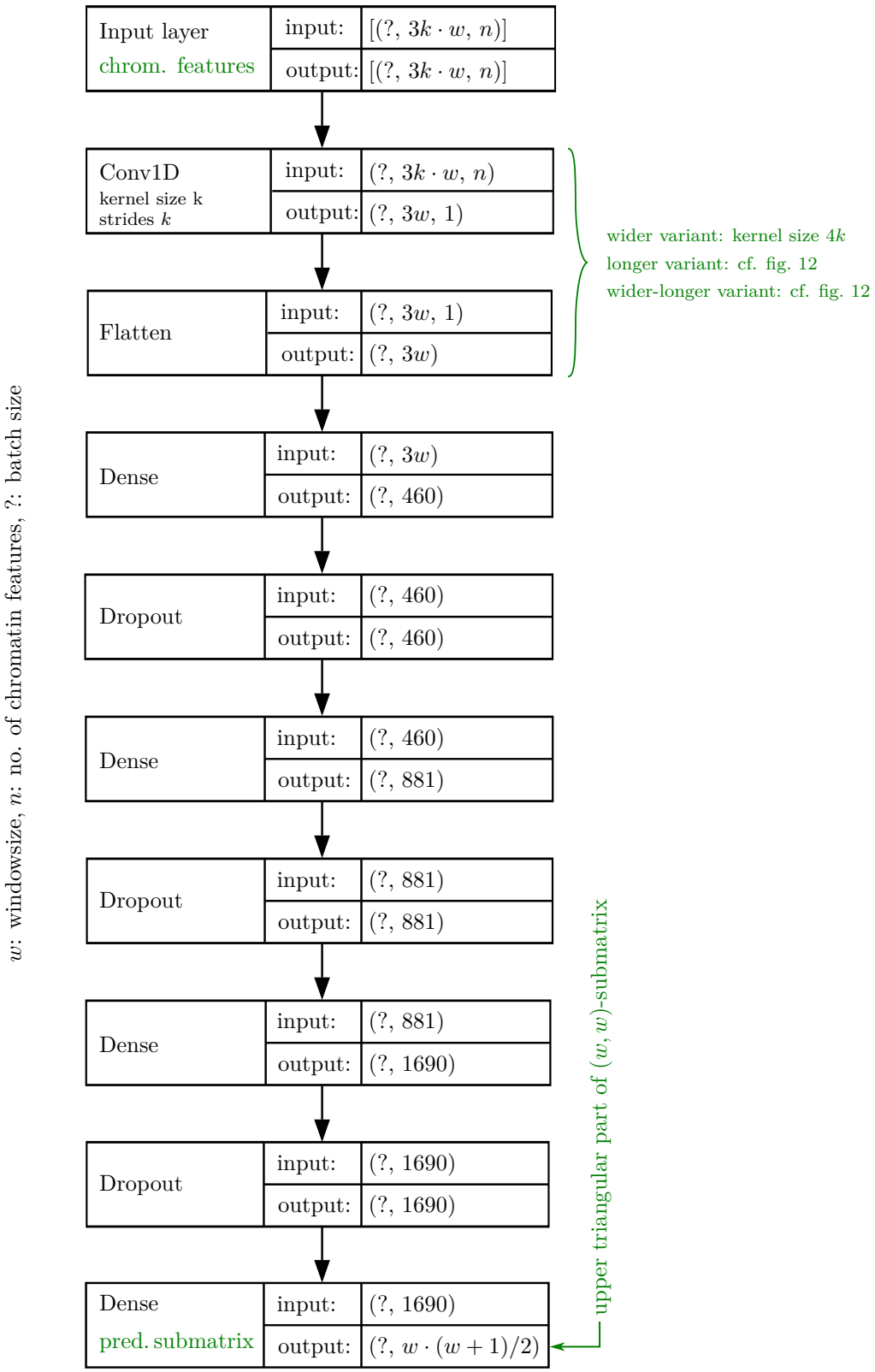


Figure 11: Basic dense neural network with generalized feature binning

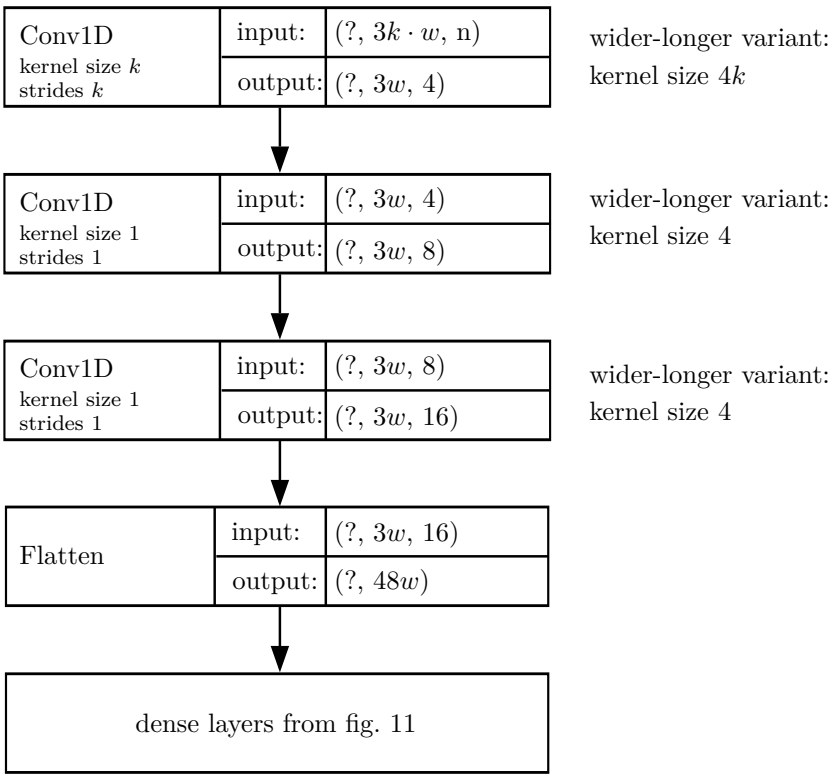


Figure 12: “Longer” and “wider-longer” variants for DNN

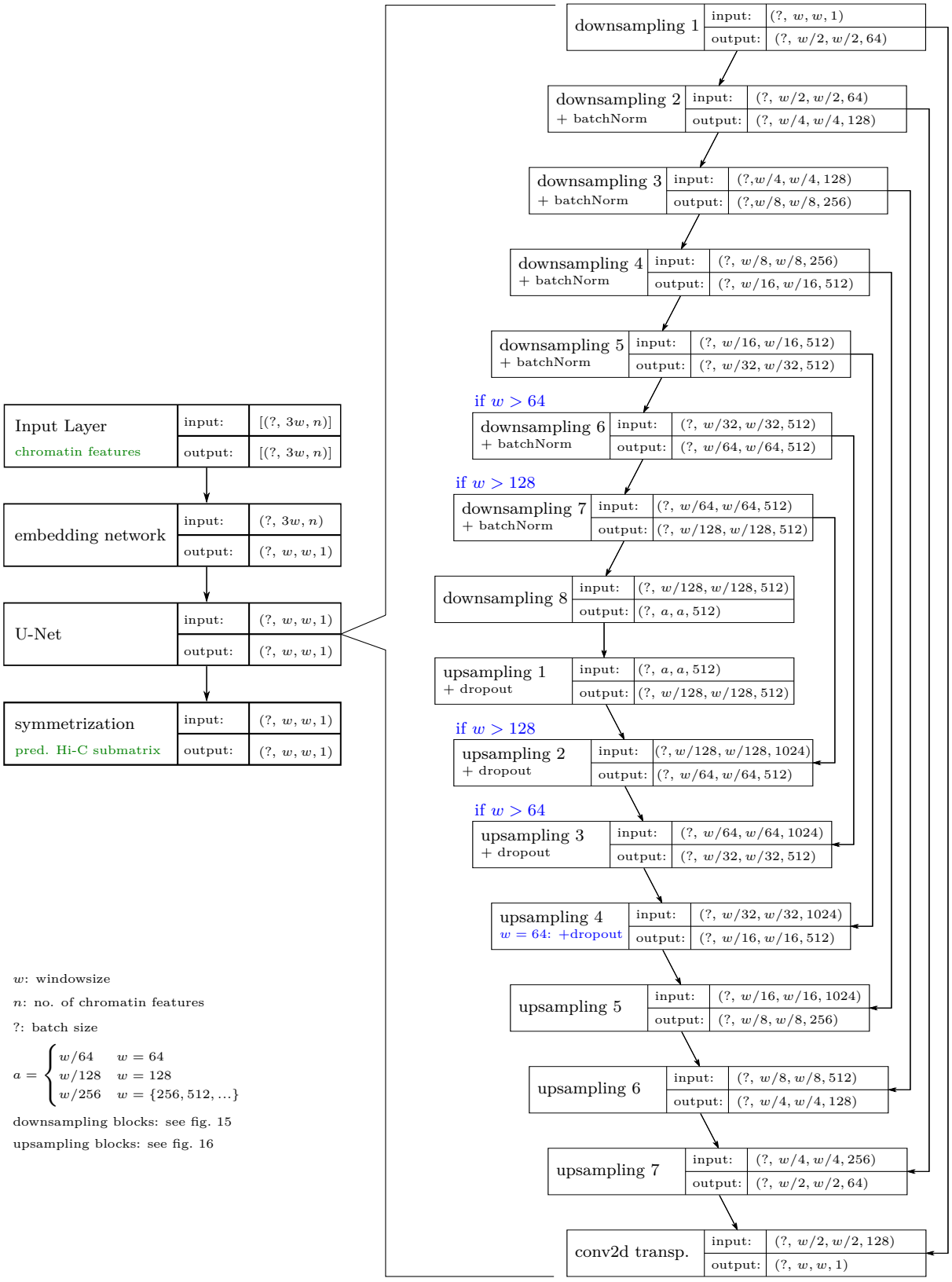


Figure 13: Adapted generator model from *pix2pix*

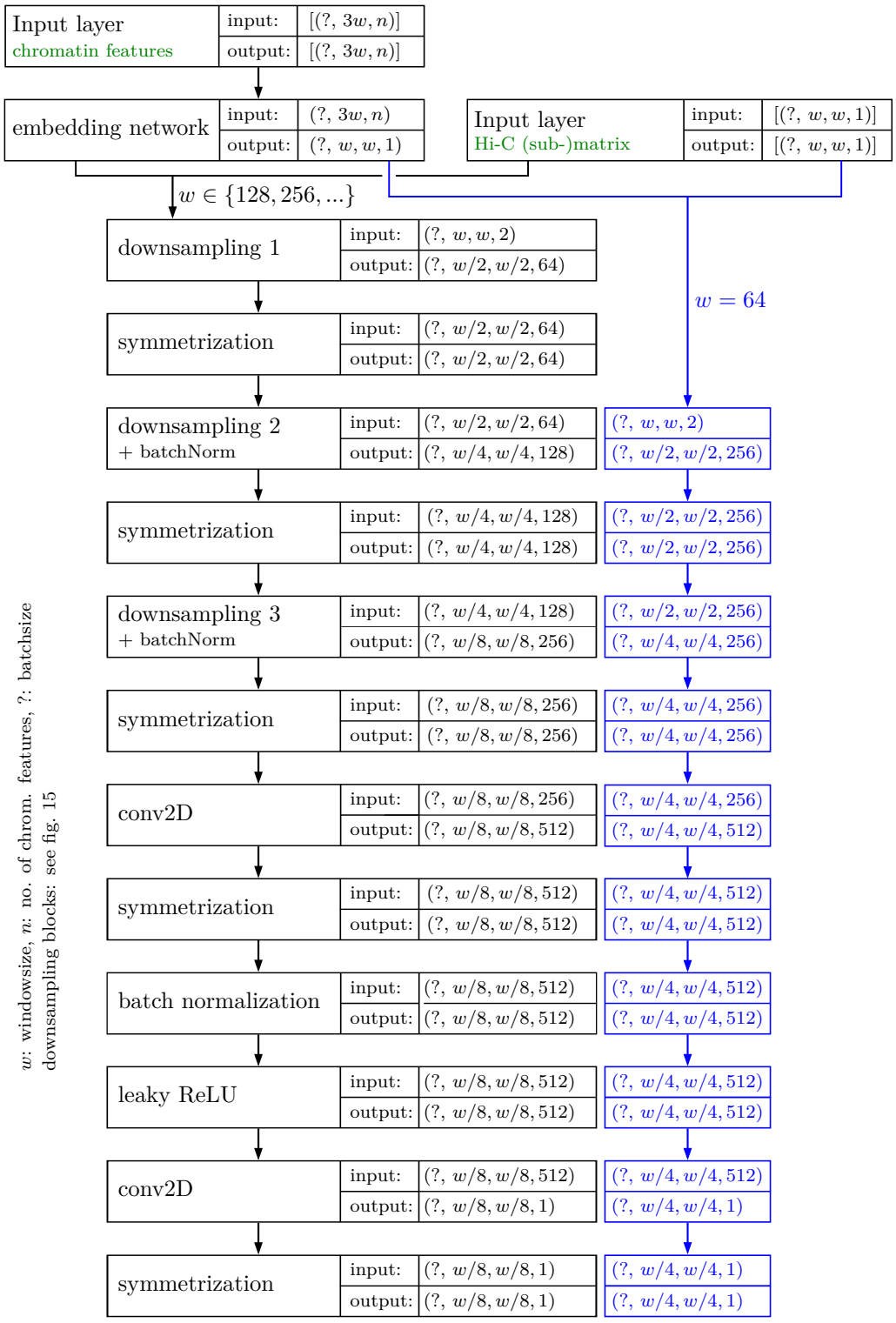


Figure 14: Adapted discriminator model from *pix2pix*

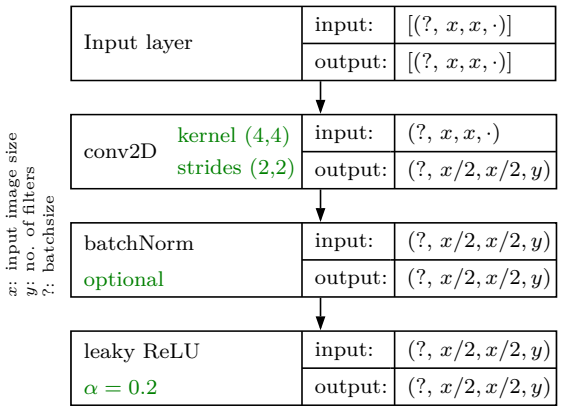


Figure 15: Downsampling block

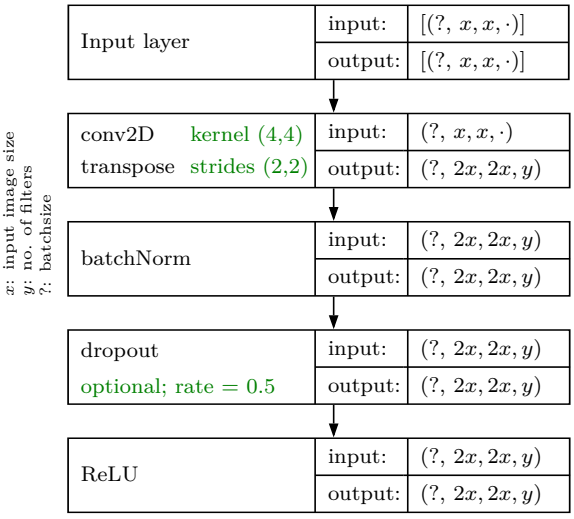


Figure 16: Upsampling block

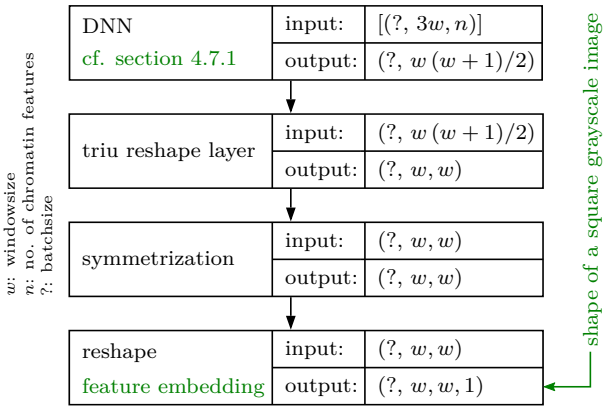


Figure 17: Embedding network, DNN

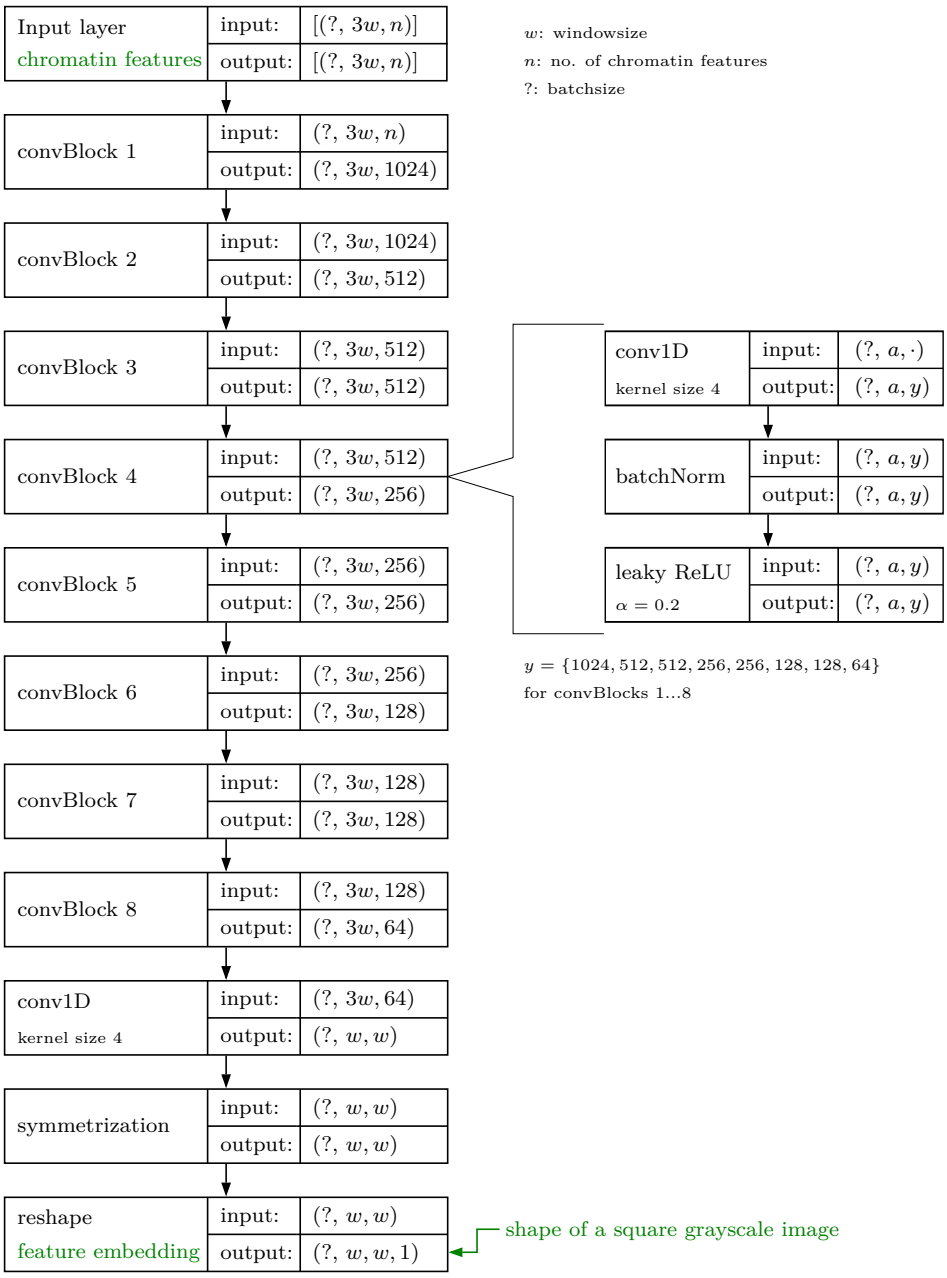


Figure 18: Embedding network, CNN

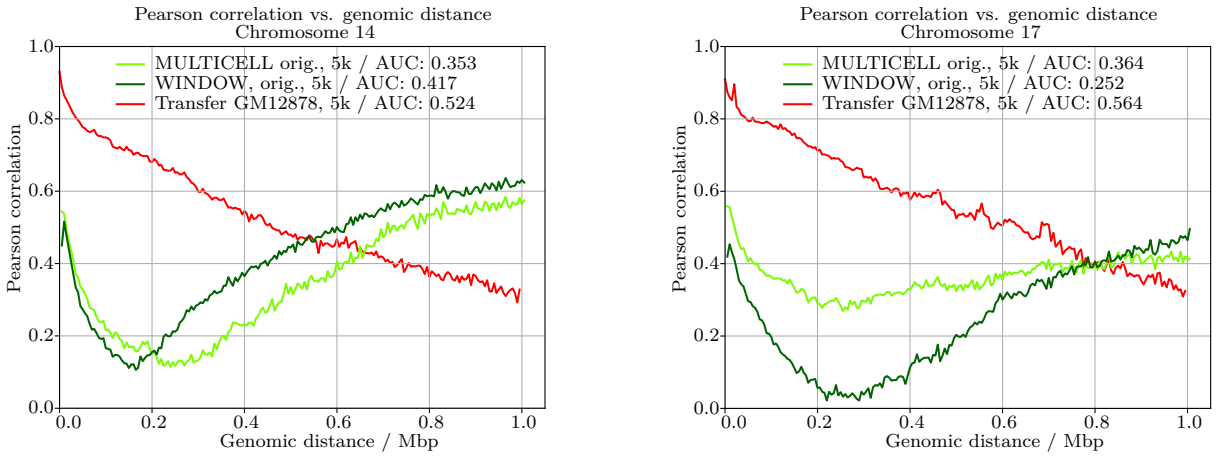


Figure 19: Pearson correlations reconstructed from [28]

5 Results

In the following two sections, the results of the DNN- and cGAN-based approaches for predicting Hi-C contact matrices will be presented. While the various modifications of the dense neural network described in section 3.1 did not really help improve the status quo, the novel conditional generative adversarial network laid out in section 3.2 showed interesting properties.

5.1 Dense Neural Network approaches

In the following subsections, the resulting predictions for modifications of the dense neural network originally conceived by Farré, Heurteau, Cuvier and Emberly [30] will be shown. This includes variations of the convolutional layer(s), custom loss functions with the intent of reducing blurriness in the predictions and tuning window- and bin size, cf. section 3.1. As a start, however, the results of the initial network without any modifications will be shown for comparison.

5.1.1 Initial results for comparison

The basic network was setup and trained as explained in section 4.7.1. Here, the validation error (MSE) reached its minimum of about 150 000 after approximately 500 epochs for 25 kbp bin size and around 400 epochs for 10 kbp bin size, fig. 20f and 21f. Beyond that, the learning curve indicated overfitting, but the resulting test matrices often still looked fairly similar, compare e. g. the matrix plots after 500 and 1000 epochs in figures 22 and 23.

Figure 20 and fig. 21 show the distance stratified Pearson correlations (cf. section 4.5) alongside area under the correlation curve (AUC) for the five test chromosomes at bin sizes 25 and 10 kbp, respectively. The red curves in each correlation plot show the correlation between the corresponding training chromosome from GM12878 and the target chromosome from K562. It is obvious that all predicted test matrices have a strictly positive Pearson correlation with respect to the experimentally derived matrices by Rao et al. [11], but are not better than simply taking data from the training cell line as prediction for the target cell line.

The predicted matrices themselves looked modest when plotted with pygenometracks. While the DNN generally produced high interaction counts in regions with many true interactions and low interaction counts in regions with few true interactions, the (TAD-)boundaries between different interacting domains were mostly not discernible, fig. 22 and 23. This finding is in line with the clearly positive, but medium-valued Pearson correlations. Exceptions with more distinct boundaries existed in all of the five test chromosomes, for example chr19, 34 to 35 kbp (fig. 22b), but were rare. Interestingly, medium-sized interacting structures, for example chr21, 31 to 32.5 kbp or between chr19, 31.2 to 32.7 kbp often seemed to be missing altogether – while structures larger than the window size, for example chr3, 34 and 36.7 kbp and 36.7 and 39.5 kbp sometimes were at least indicated.

Reducing the bin size to $b_{feat} = b_{mat} = 10\,\text{kbp}$ as in the paper by Farré et al. [30] led to somewhat different results. The area under the correlation curves was approximately the same for test chromosomes 3 and 5, slightly better for chromosome 10, but worse for chromosome 19 and 21, cf. fig. 20 and 21. However, the ability to predict larger structures was lost, and

the matrix plots thus did not look better than the ones for bin size 25 kbp. The comparatively bad result for test chromosome 21 might result from the low chromatin feature coverage of this particular chromosome.

No obvious correlation between comparatively "good" and "bad" predictions with open and closed states of the chromatin was observed. However, formally computing such a correlation is challenging, because no adequate objective measure for "good" and "bad" is known, especially considering the rather blurry results obtained so far. Furthermore, even if suchlike correlations existed, exploiting them for improving predictions would still be, at best, not straightforward.

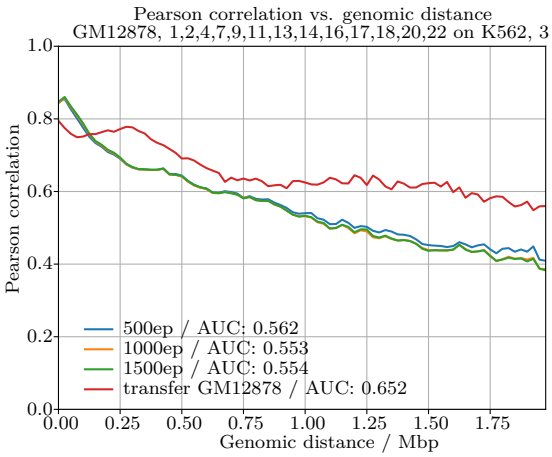
5.1.2 Results for variations of the convolutional part

The results for the "wider" network, which featured a wider convolutional filter in the first network layer, cf. section 3.1.2 and 4.7.2, were generally similar to the initial results, both in terms of Pearson correlations and in terms of matrix plots, fig. 25 and 26. Given the small increase in the number of trainable parameters and overall similar network topology, this is not surprising. Overfitting was less obvious than with the initial setup and the training process looked more smooth, but the remaining validation error was slightly higher than for the initial approach, fig. 25f.

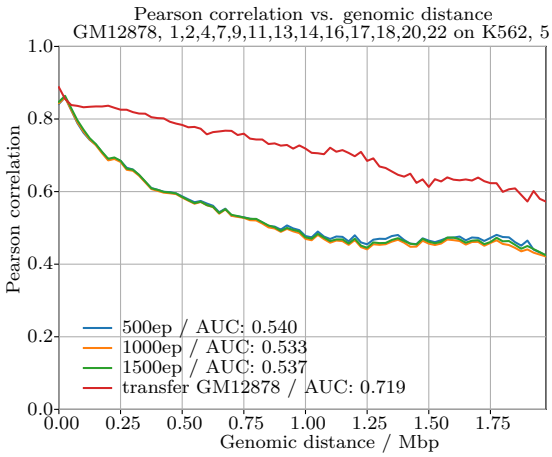
The predictions from the "longer" variant with three convolutional filter layers instead of a single one were better than the initial predictions in terms of Pearson correlations for test chromosomes 10, 19 and 21, but worse for test chromosomes 3 and 5, fig. 27. Interestingly, correlations for some of the larger distances could not be computed after 250 and 500 epochs, which generally means that the same values were predicted for these distances, cf. section 4.5. The reason for this behavior is not fully understood yet, but comparatively few neurons in the outermost layer are responsible for predictions at longer distances due to the chosen network setup, cf. section 4.2, fig. 9. Since the longer network variant has a considerably larger number of trainable parameters, it is assumed that 500 epochs might not have been enough to activate some of the outer neurons. Slow training can occur when ReLU activations are used (as in the given case) and the gradients are close to zero [69]. Apart from that, the learning process for the "longer" variant in general looked more smooth and reached a lower validation error than before, fig. 27f, but the matrix plots did not show any obvious improvement over the initial ones, fig. 28.

Combining the "longer" and "wider" variants in the "wider-longer" setup with more convolutional layers and wider filter kernels also did not perform as expected. While improvements in the Pearson correlations could again be seen for 3 of 5 test chromosomes compared to the initial network, fig. 29, the observed correlations were worse than the ones from the highly similar "longer"-variant alone. Like with the similar "longer"-approach predictions at longer distances were partially missing. Compared to the other variants, the validation error was generally higher and stopped decreasing after very few epochs, fig. 29f. In terms of matrix plots, the predictions surprisingly were still quite similar to the initial ones, but seemed a bit more blurry, 30.

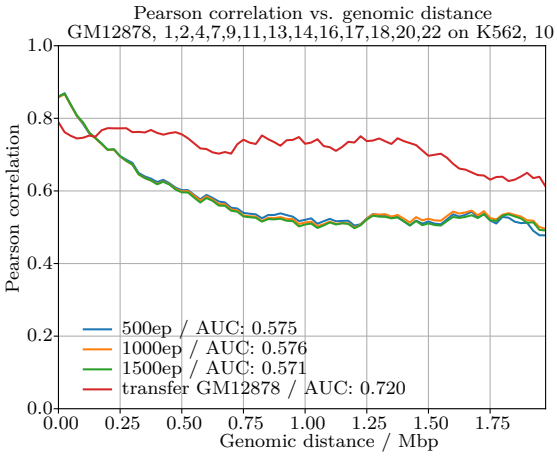
Predictions and metrics from the generalized DNN approach with feature bin size 5 kbp and matrix bin size 25 kbp are shown in fig. 31 and 32. Unfortunately, the results did again not improve compared to the initial predictions. While the learning curve was smooth and showed signs of slight overfitting beyond 300 epochs, fig. 31f, the matrix plots seemed worse than the



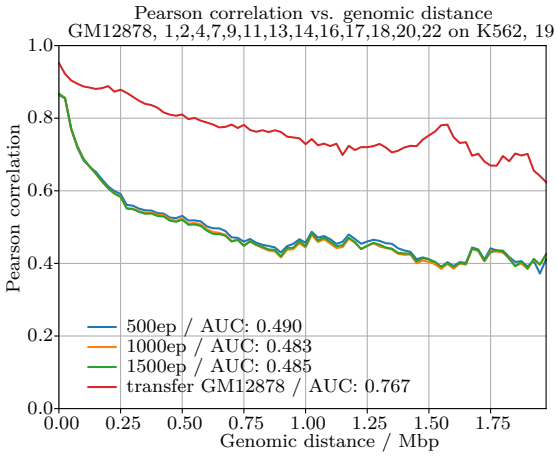
(a) chr3



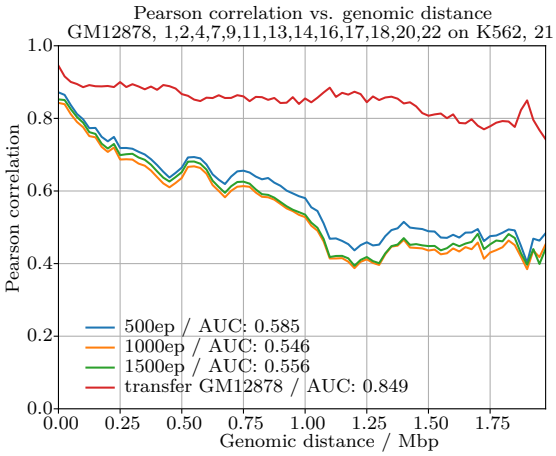
(b) chr5



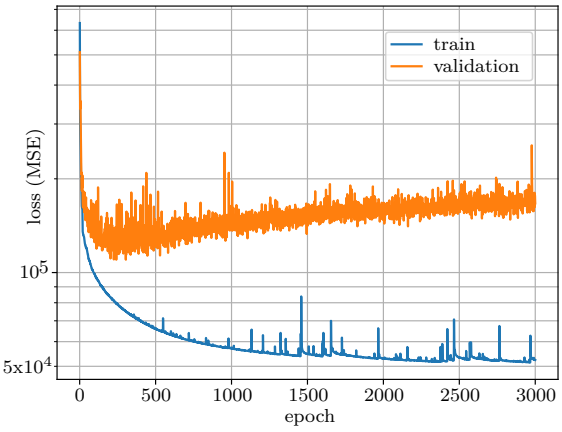
(c) chr10



(d) chr19

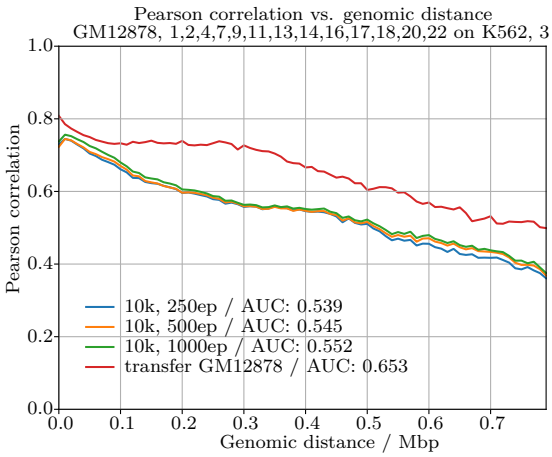


(e) chr21

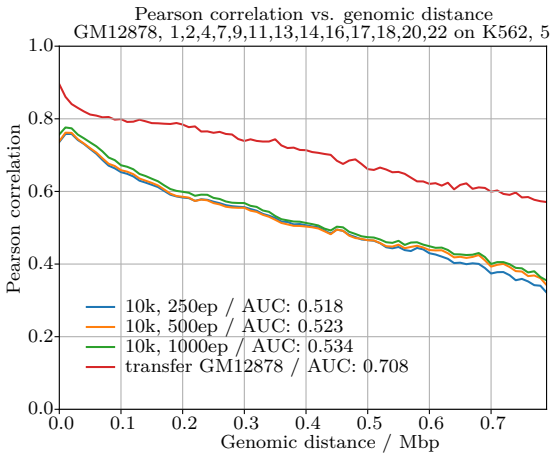


(f) Learning progress

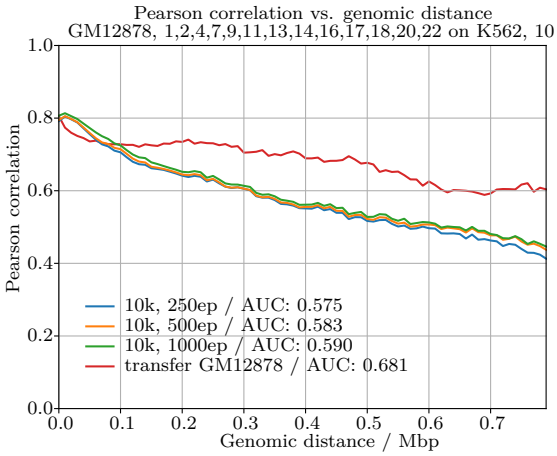
Figure 20: Results / metrics, basic DNN, 25 kbp, test chromosomes



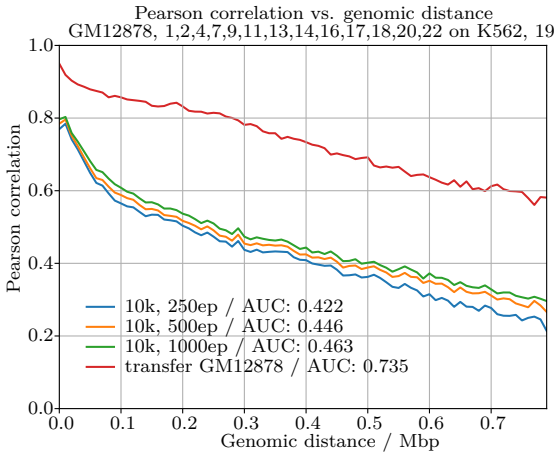
(a) chr3



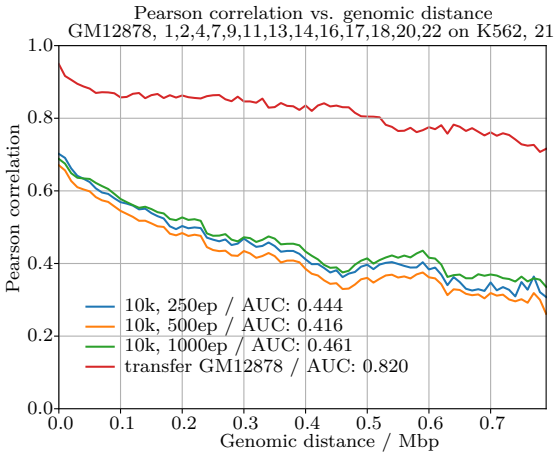
(b) chr5



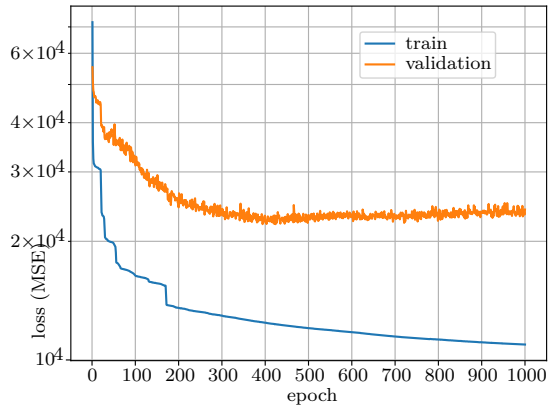
(c) chr10



(d) chr19

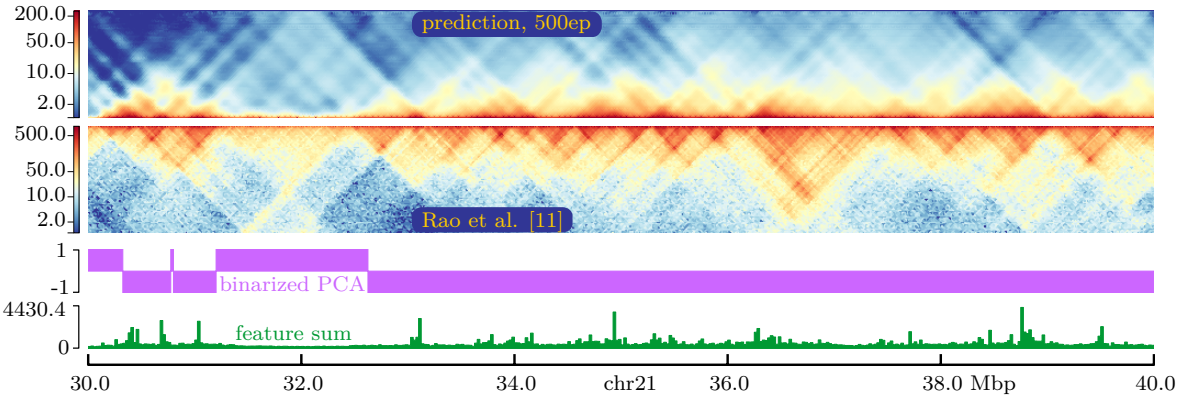


(e) chr21



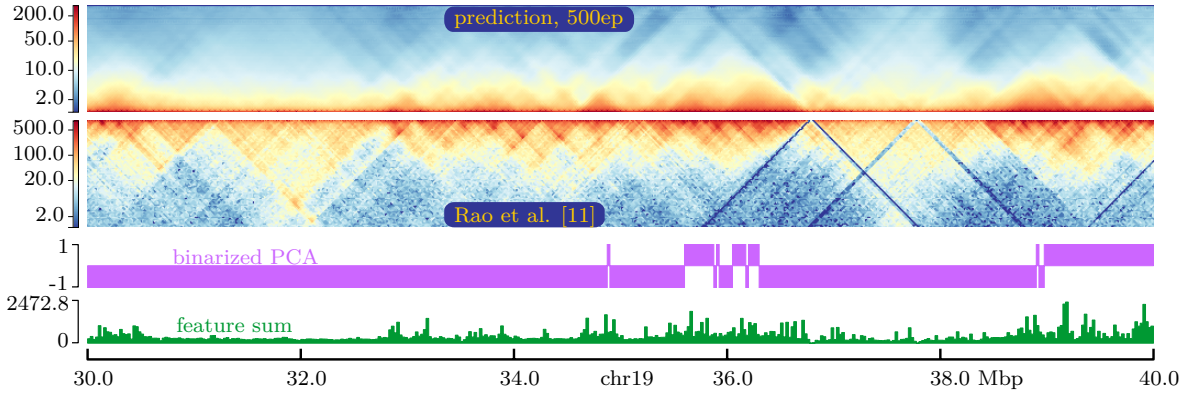
(f) Learning progress

Figure 21: Results / metrics, basic DNN, 10 kbp, test chromosomes



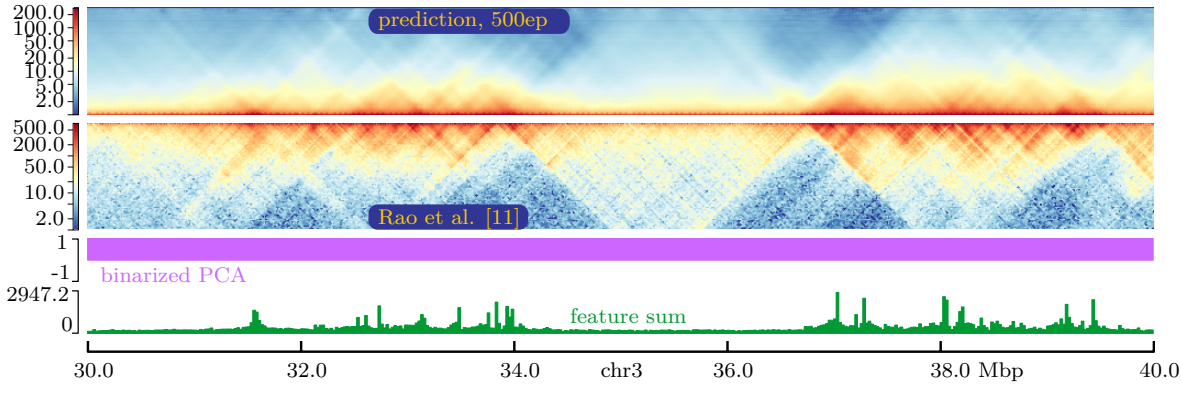
(a) example region 1, 500 epochs

see section 4.6 for track explanations



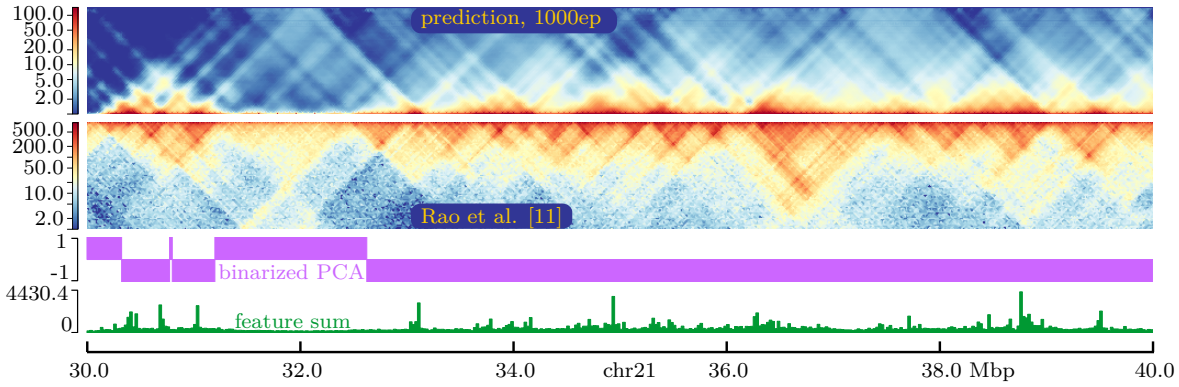
(b) example region 2, 500 epochs

see section 4.6 for track explanations



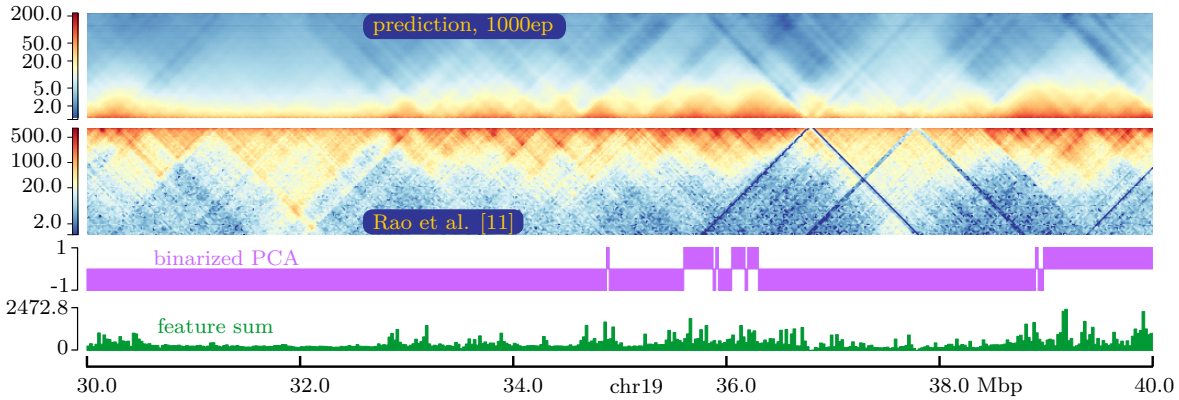
(c) example region 2, 500 epochs

Figure 22: Example predictions GM12878 → K562, basic DNN, 25 kbp, 500 epochs



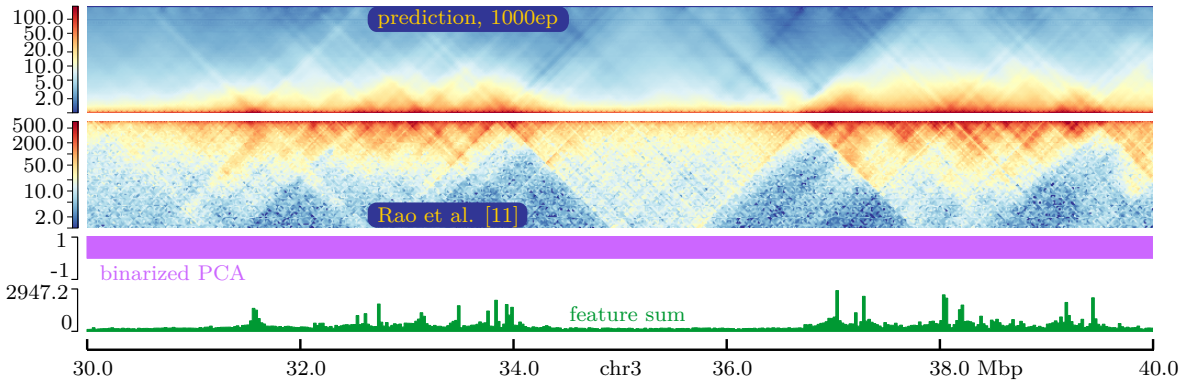
(a) Example region 1, 1000 epochs

see section 4.6 for track explanations



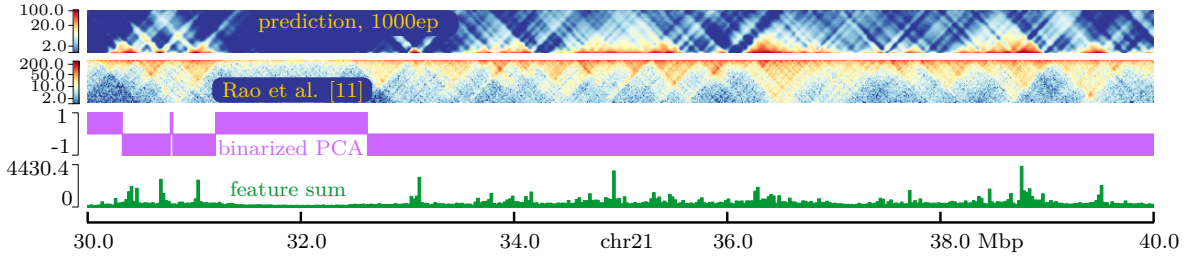
(b) Example region 2, 1000 epochs

see section 4.6 for track explanations



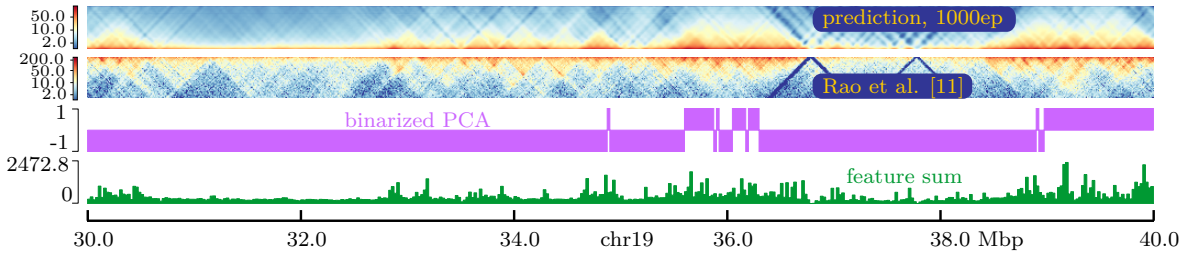
(c) Example region 2, 1000 epochs

Figure 23: Example predictions GM12878 → K562, basic DNN, 25 kbp, 1000 epochs



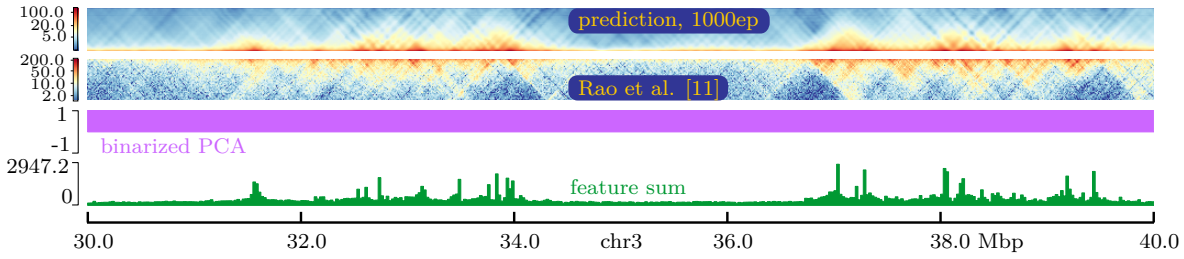
(a) Example region 1

see section 4.6 for track explanations



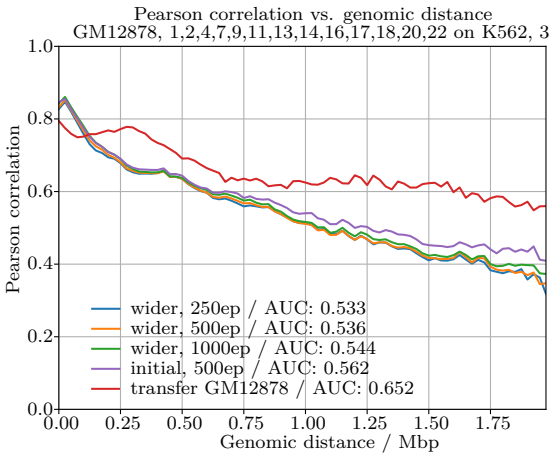
(b) Example region 2

see section 4.6 for track explanations

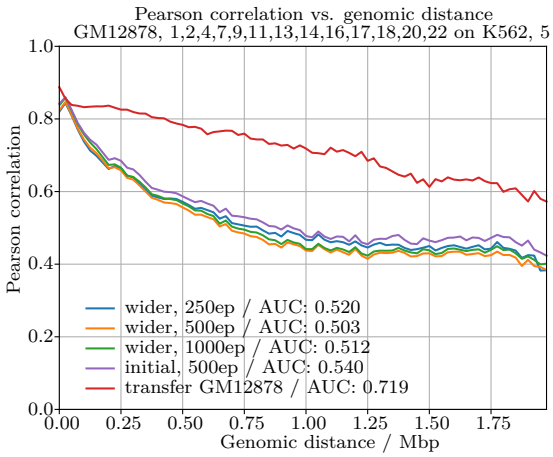


(c) Example region 3

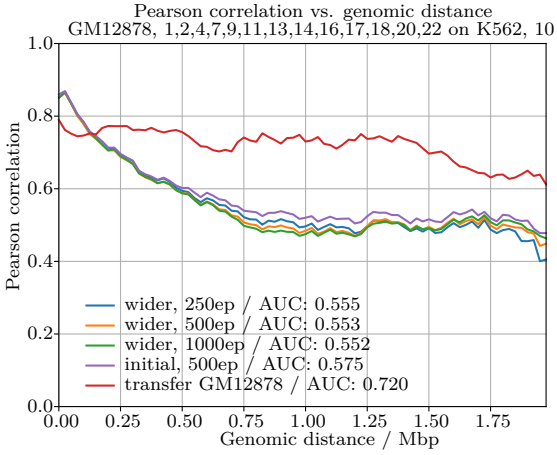
Figure 24: Example predictions GM12878 → K562, basic DNN 10 kbp, 1000 epochs



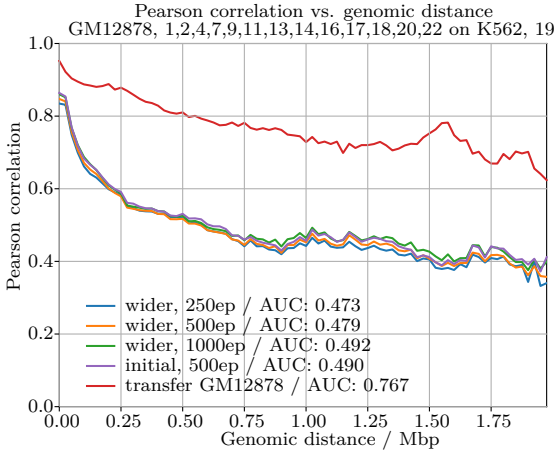
(a) chr3



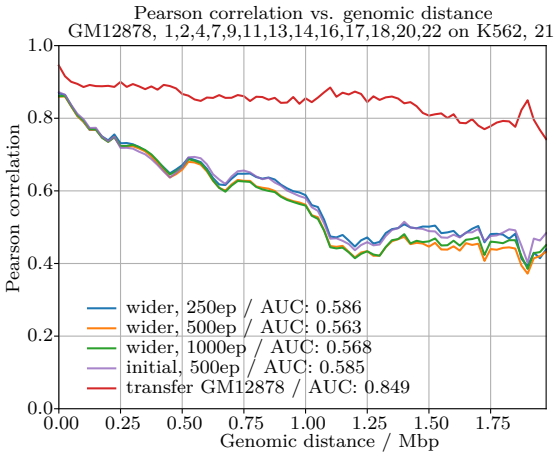
(b) chr5



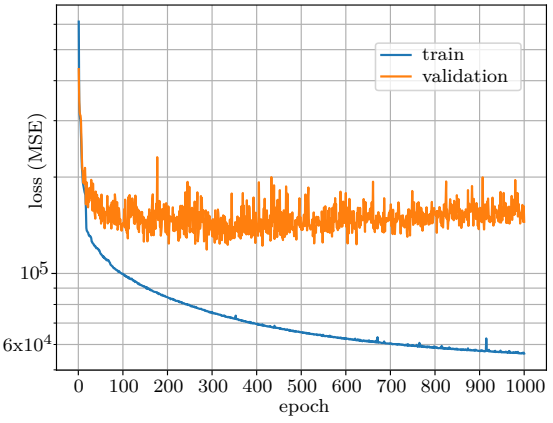
(c) chr10



(d) chr19

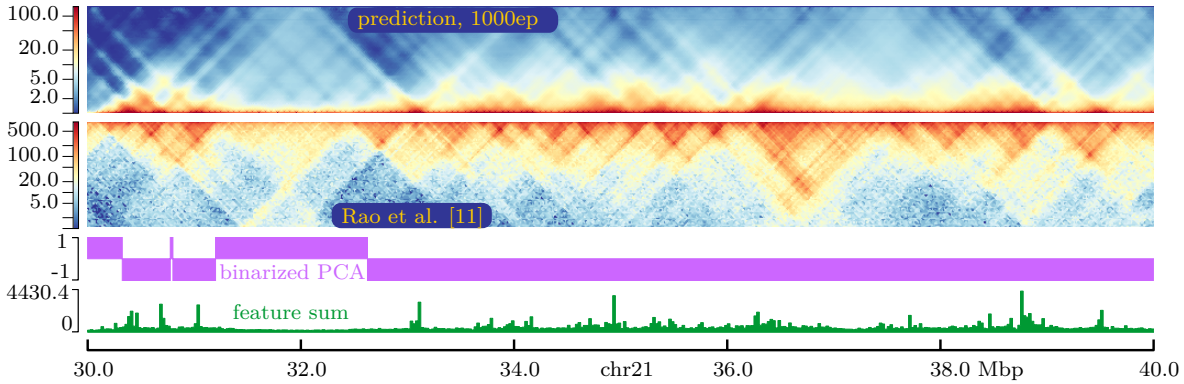


(e) chr21



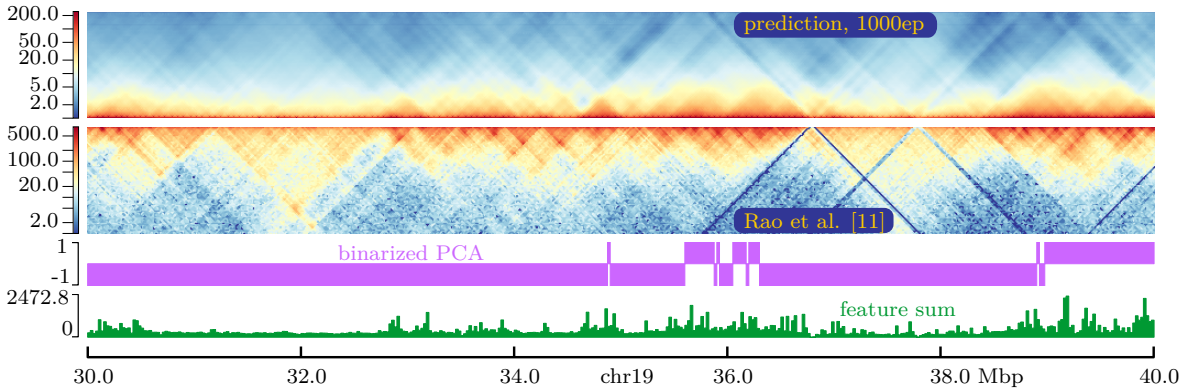
(f) Learning progress for wider DNN

Figure 25: Pearson correlations, “wider” variant of DNN, test chromosomes



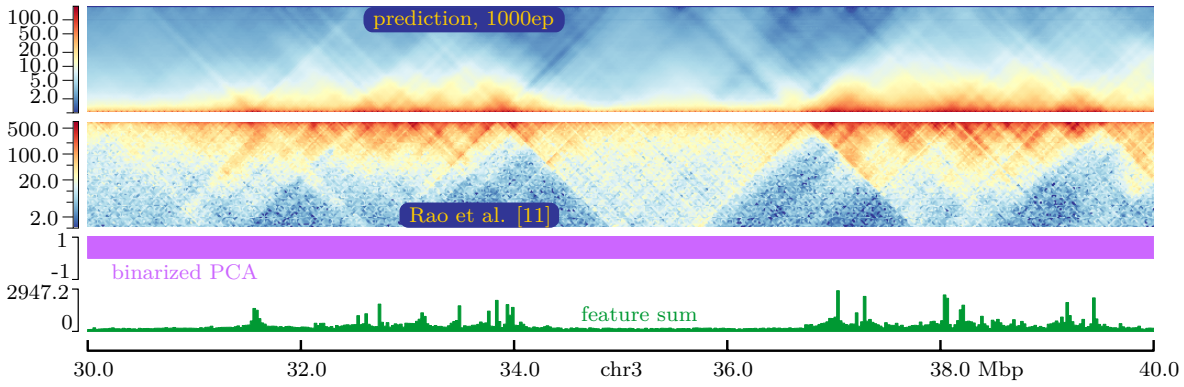
(a) Example region 1

see section 4.6 for track explanations



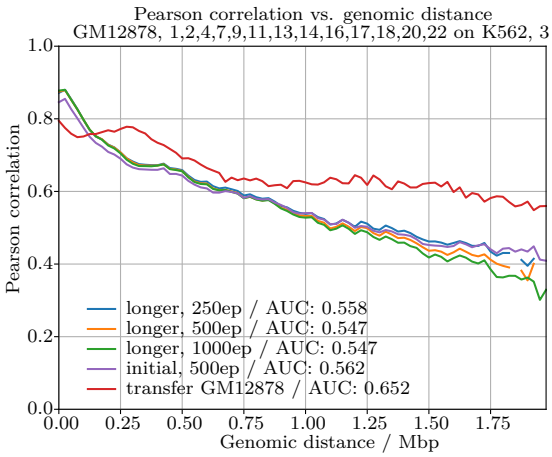
(b) Example region 2

see section 4.6 for track explanations

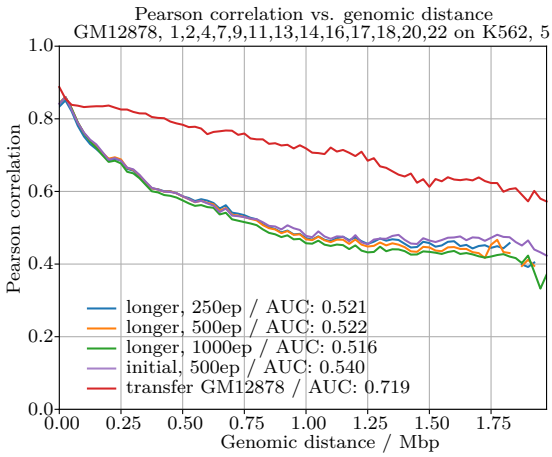


(c) Example region 3

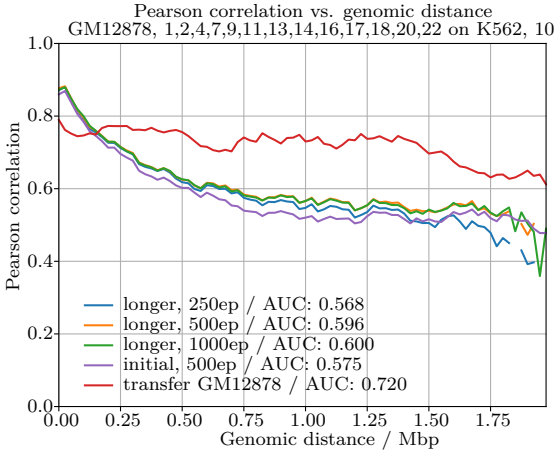
Figure 26: Example predictions GM12878 → K562, "wider" variant of DNN 25 kbp, 1000 epochs



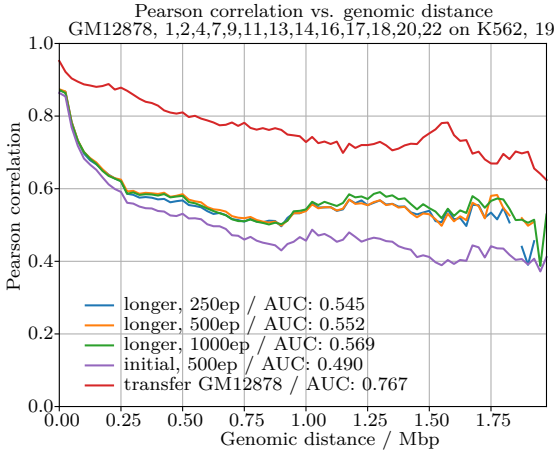
(a) chr3



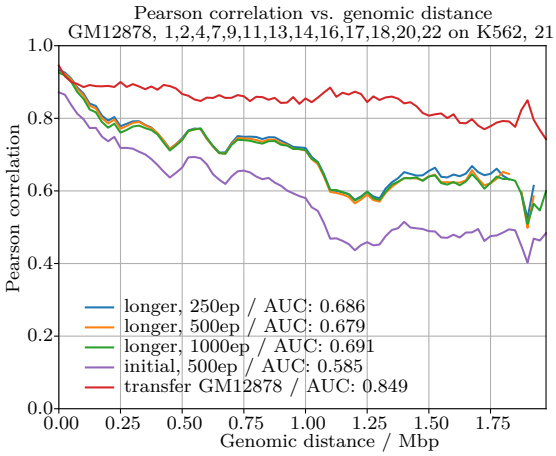
(b) chr5



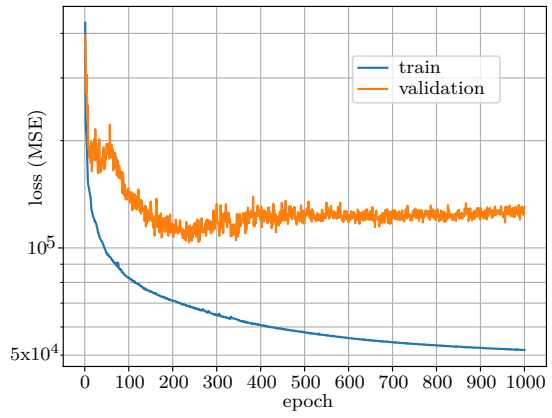
(c) chr10



(d) chr19

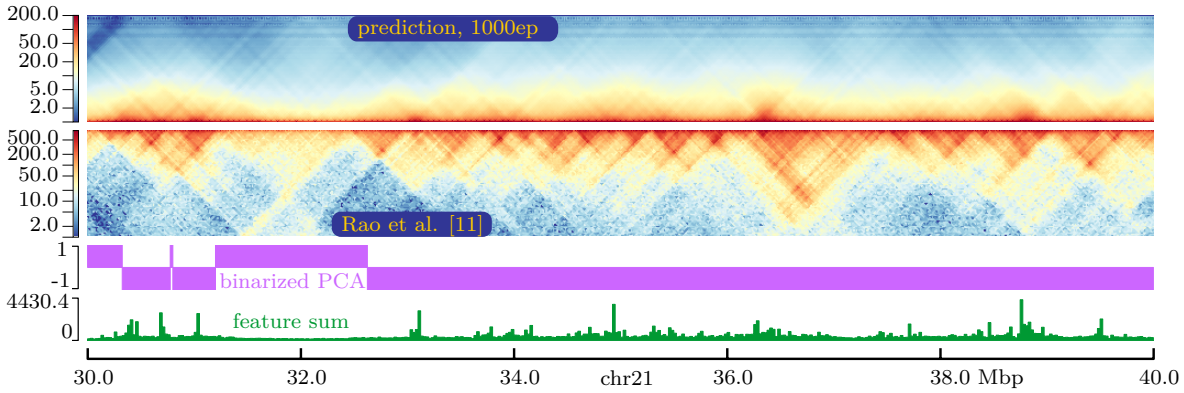


(e) chr21



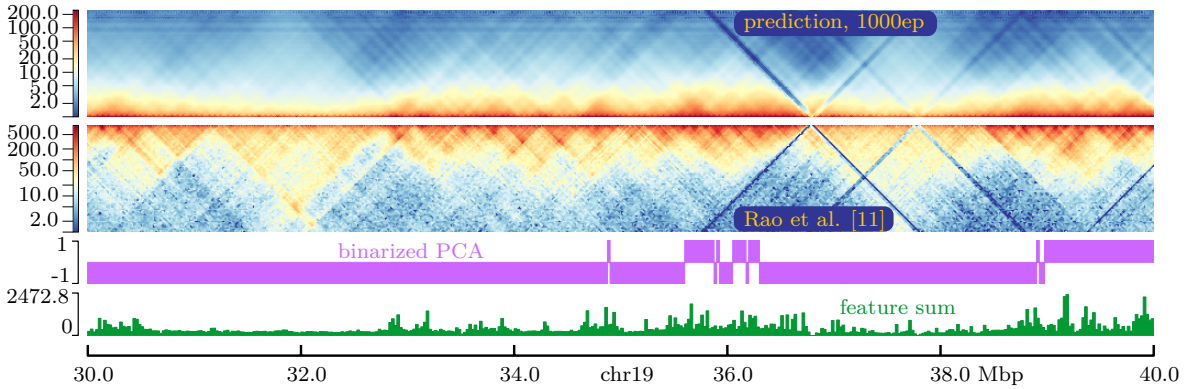
(f) Learning progress for longer DNN

Figure 27: Results / metrics, “longer” variant of DNN, test chromosomes



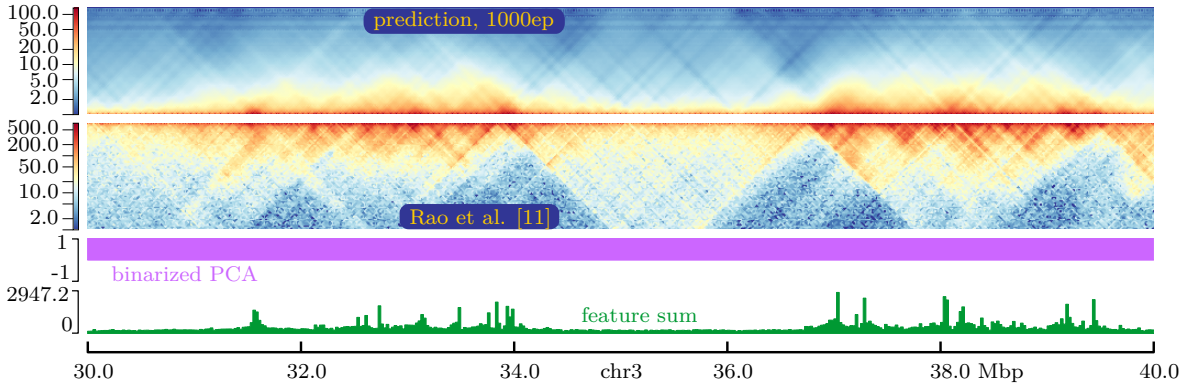
(a) Example region 1

see section 4.6 for track explanations



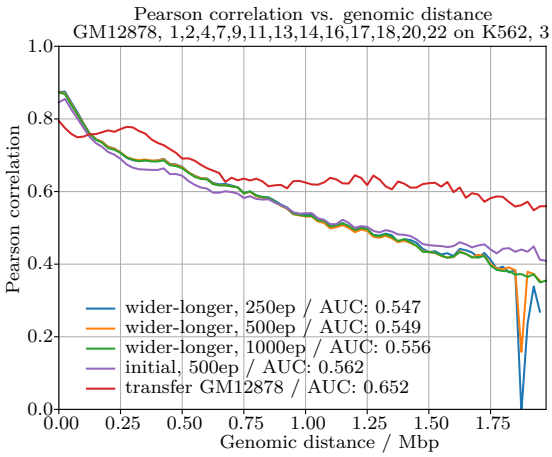
(b) Example region 2

see section 4.6 for track explanations

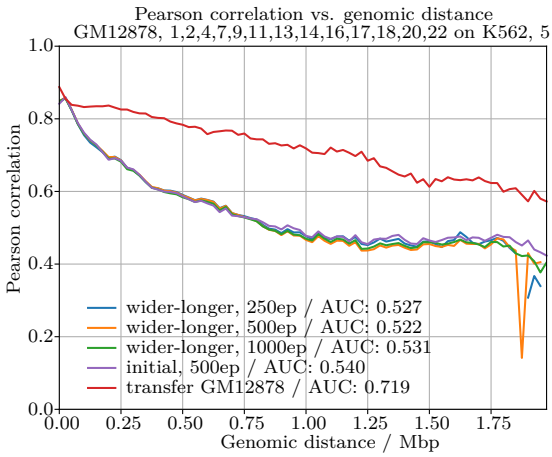


(c) Example region 3

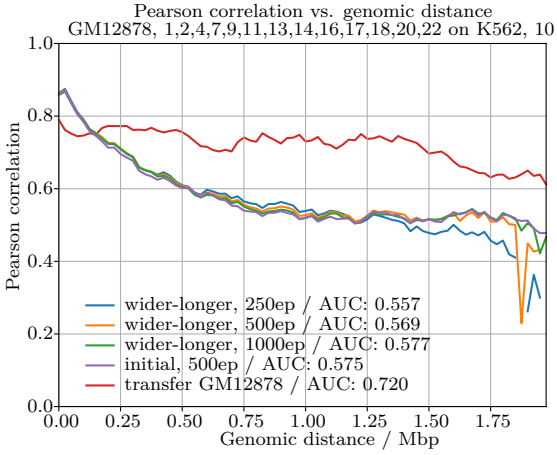
Figure 28: Example predictions GM12878 → K562, "longer" variant of DNN 25 kbp, 1000 epochs



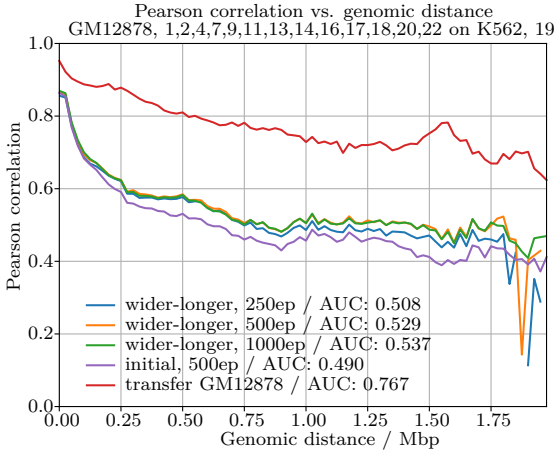
(a) chr3



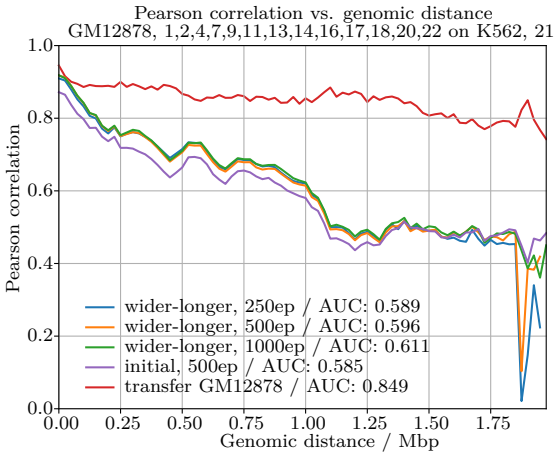
(b) chr5



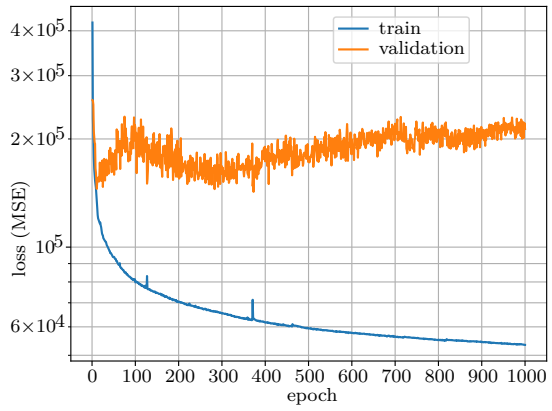
(c) chr10



(d) chr19

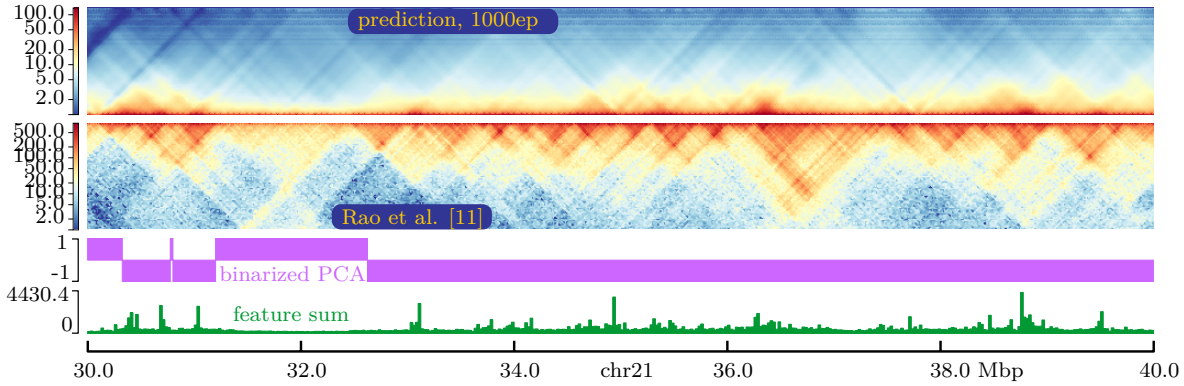


(e) chr21



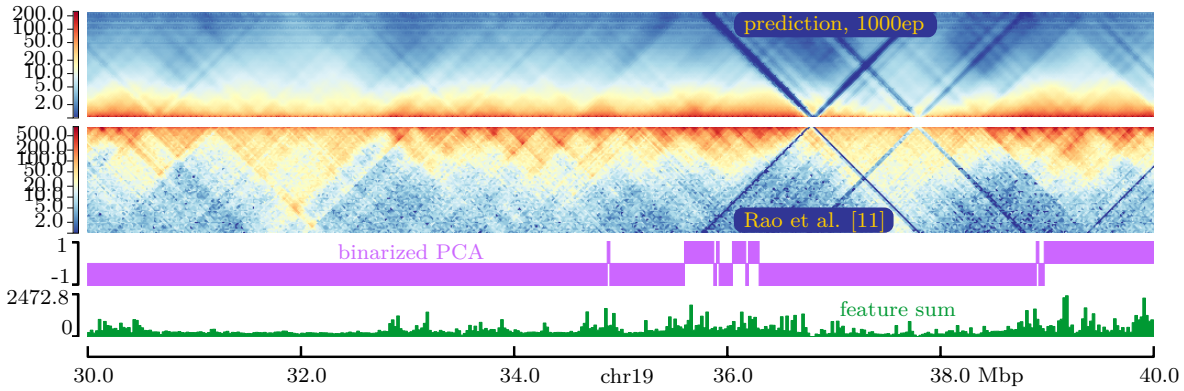
(f) Learning progress

Figure 29: Results / metrics, "wider-longer" variant of DNN, test chromosomes



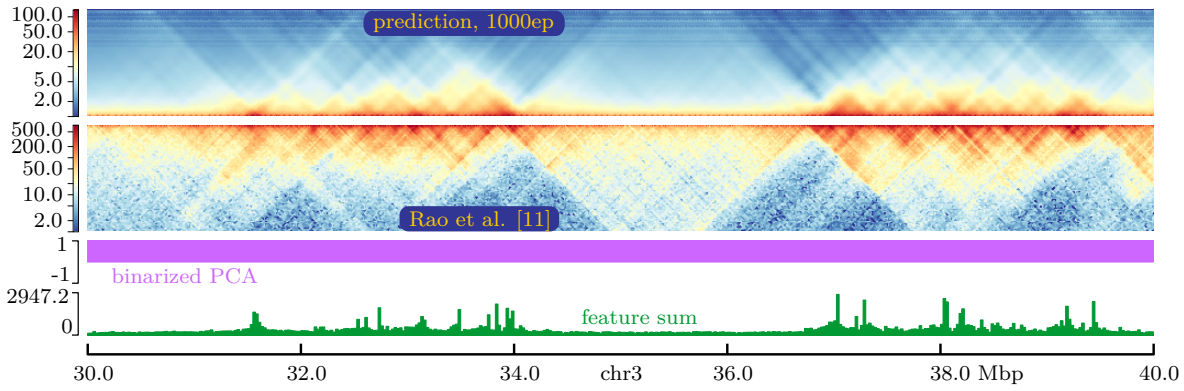
(a) Example region 1

see section 4.6 for track explanations



(b) Example region 2

see section 4.6 for track explanations



(c) Example region 3

Figure 30: Example predictions GM12878 → K562, "wider-longer" variant of DNN, 25 kbp, 1000 epochs

initial ones. For example, the large structure at chromosome 3, 34 to 36.7 kbp, which had been detected by the previous approaches, was completely missing.

5.1.3 Results for combined loss function

Exchanging the mean squared error for a combined loss function consisting of MSE, TV loss and perceptual loss did not improve the results in the given setting. The results for optimizing a loss function according to eq. (13) with weighting parameters are shown in fig. 33 and 34.

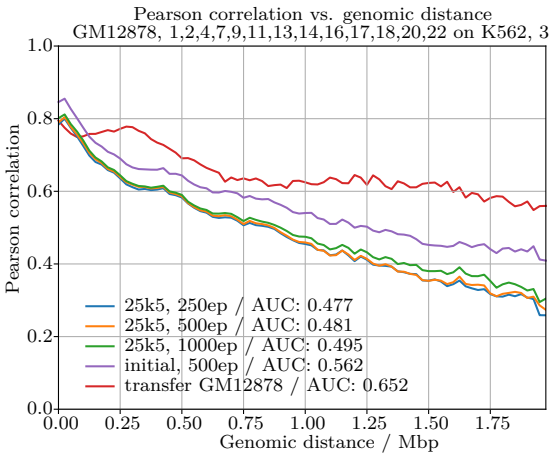
For all test chromosomes, the correlations were highly similar to the initial network's, fig. 33, and the matrix plots also looked similar, chromosome 21 probably being the most different, fig. 34. The results plotted are the best ones obtained by manual tuning of the multiplicative parameters λ. Guided parameter tuning was unfortunately infeasible within the thesis at hand due to the training times required for optimizing the combined loss function. Other options which where not explored for the same reason include truncating the *VGG-16* network at a different layer, using a loss function based on more than one of the intermediate *VGG-16* layers [54] or taking another loss network. However, the results obtained thus far were also not encouraging towards such investigations. While manually searching better parameters λ was not successful, it was found that the TV loss weight λ_{TV} needed to be much smaller than the two other weights. Otherwise, many true interactions off the matrix diagonals were considered as noise and optimized away early in the training process, cf. fig. 60 (p. 105).

5.1.4 Results for score-based loss function

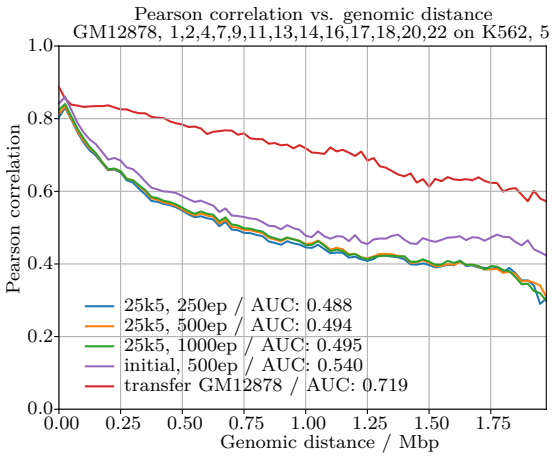
Exchanging the MSE- loss function by a combination between score-based and MSE loss allowed for a smooth learning process and a slightly lower validation error compared to the initial approach. However, at around 7 min per epoch on a GPU, the training process also was about seven times slower than the initial approach on CPU. Unfortunately, the higher effort did not lead to obvious improvements.

The Pearson correlations for a score-based loss function with parameters $\lambda_{MSE} = 1.0$, $\lambda_{score} = 100$, $ds = 12$ are shown in figure 35. While a slight improvement was achieved for test chromosome 21, the correlations of the others remained widely unchanged. The matrix plots also looked fairly similar to the initial ones, fig. 36, chromosome 21 again being the most different compared to the initial predictions.

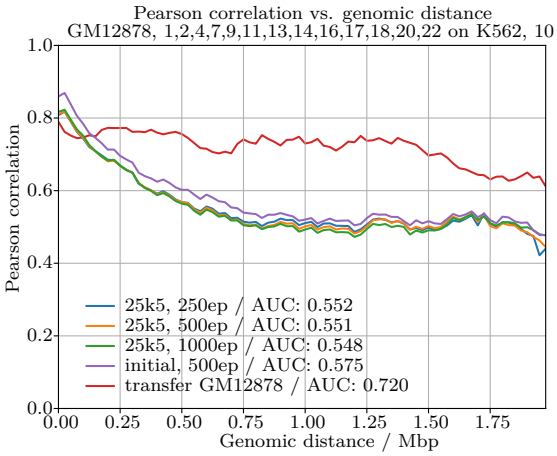
In the matrix plots, the true- and predicted scores have been added as a second track, replacing the PCA track. Indeed, the score curve computed from the true matrices showed local minima at putative TAD boundaries, as set forth in 3.1.4, so score computation with the chosen diamond size seemed sound. However, despite the optimization term in the loss function, the score curve of the predicted matrices compared to the true curve somewhat like the predicted matrices compared to the true ones: The predicted score was generally high, when the true score was high, and low when the true score was also low, but high peaks (local maxima) and steep valleys (local minima) in the plots were usually averaged out.



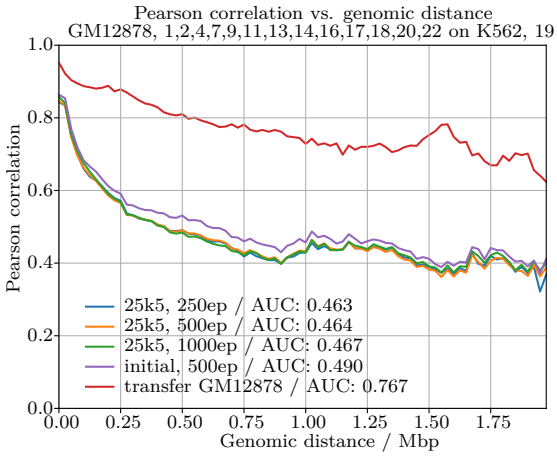
(a) chr3



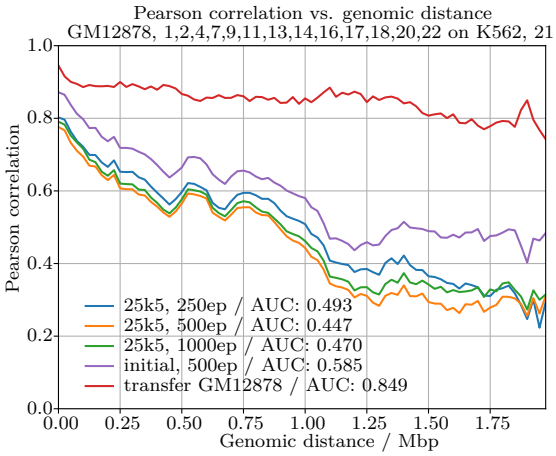
(b) chr5



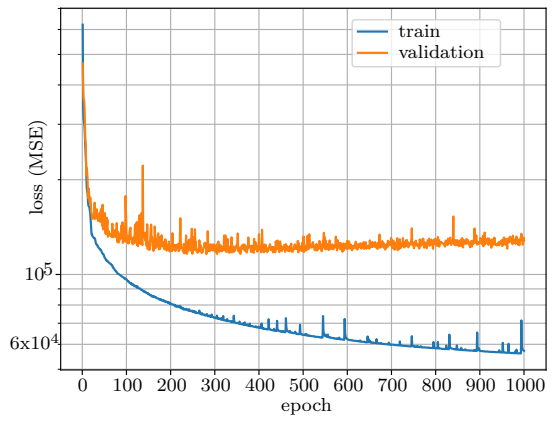
(c) chr10



(d) chr19

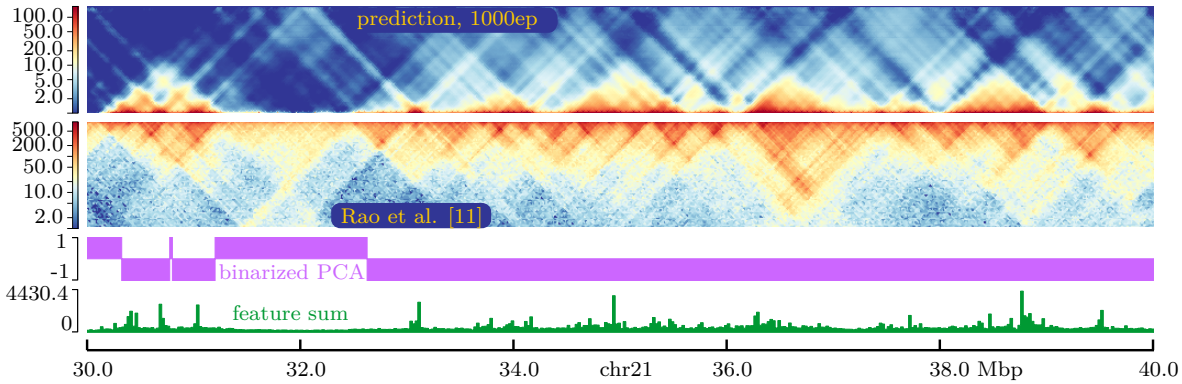


(e) chr21



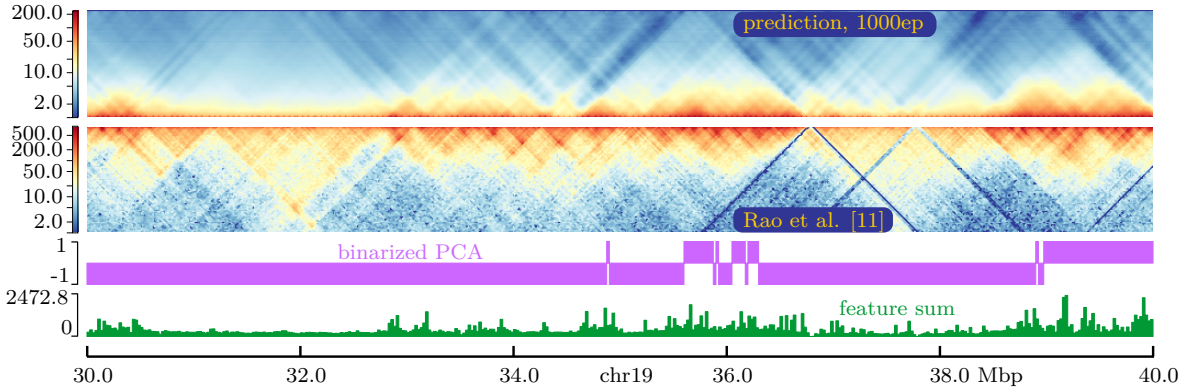
(f) Learning progress

Figure 31: Results / metrics, "5k – 25k" variant of DNN with $b_{feat} = 5\,\text{kbp}$ and $b_{mat} = 25\,\text{kbp}$, test chromosomes



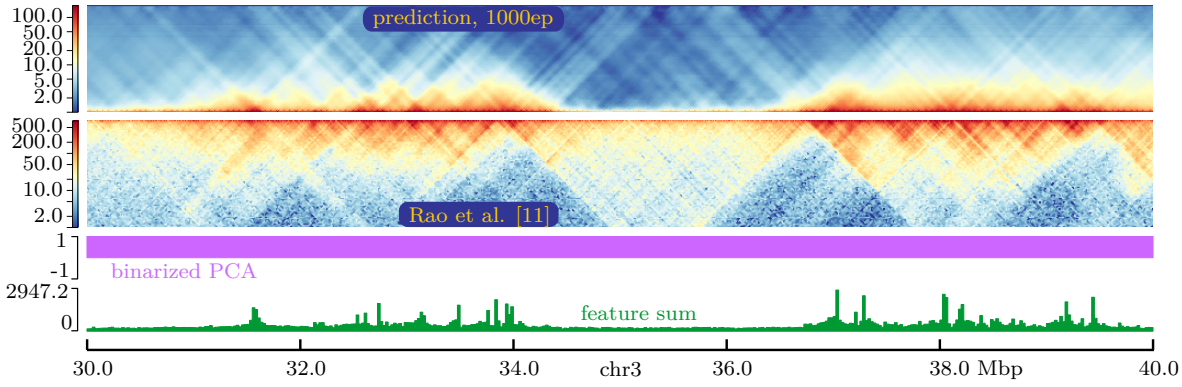
(a) Example region 1

see section 4.6 for track explanations



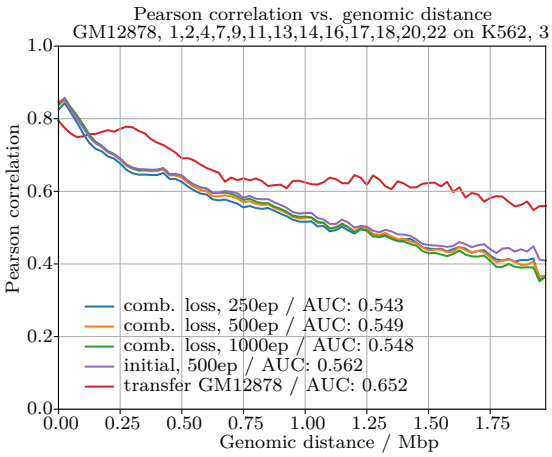
(b) Example region 2

see section 4.6 for track explanations

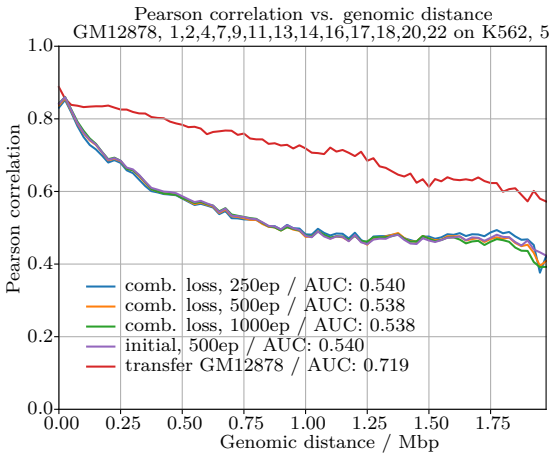


(c) Example region 3

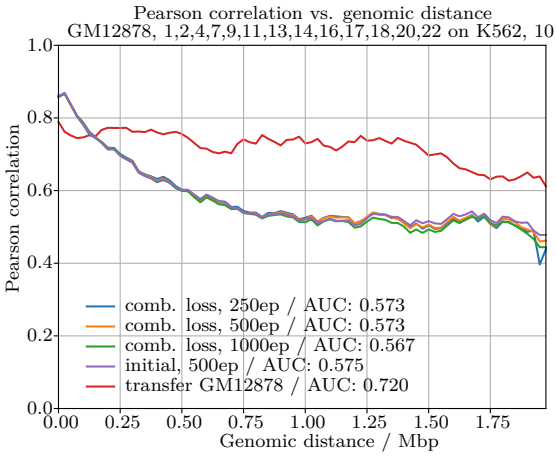
Figure 32: Example predictions GM12878 → K562, "5k – 25k" variant of DNN, 1000 epochs



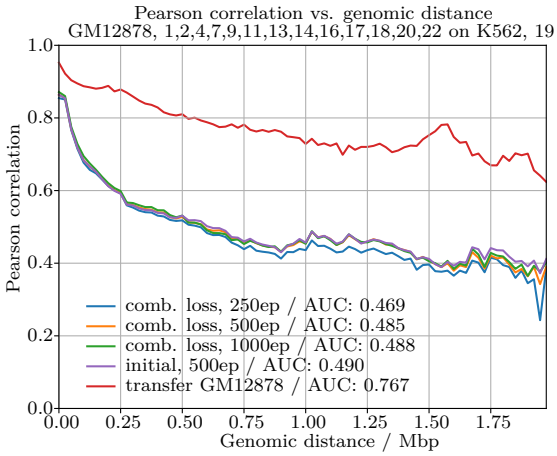
(a) chr3



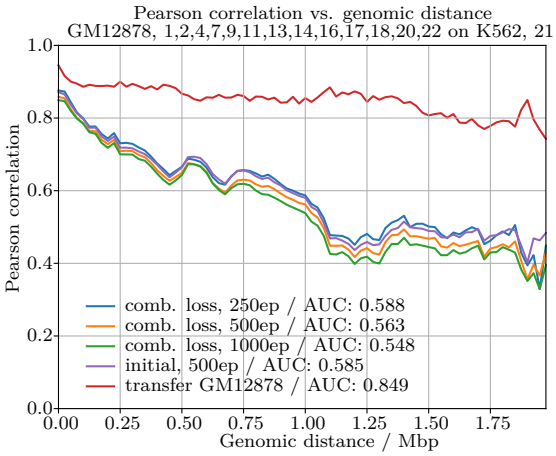
(b) chr5



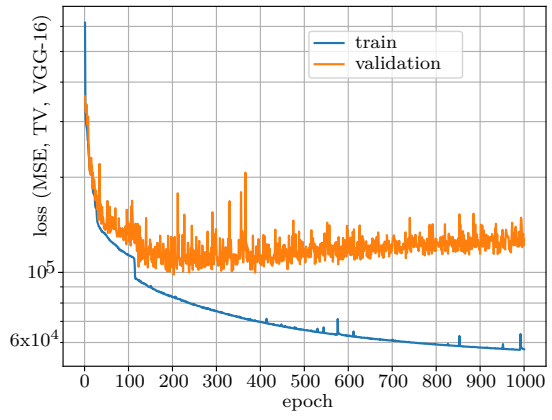
(c) chr10



(d) chr19

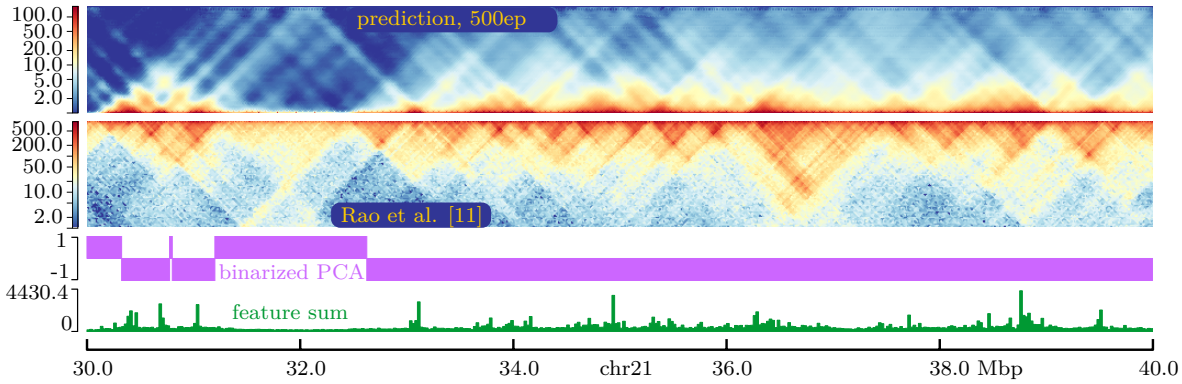


(e) chr21



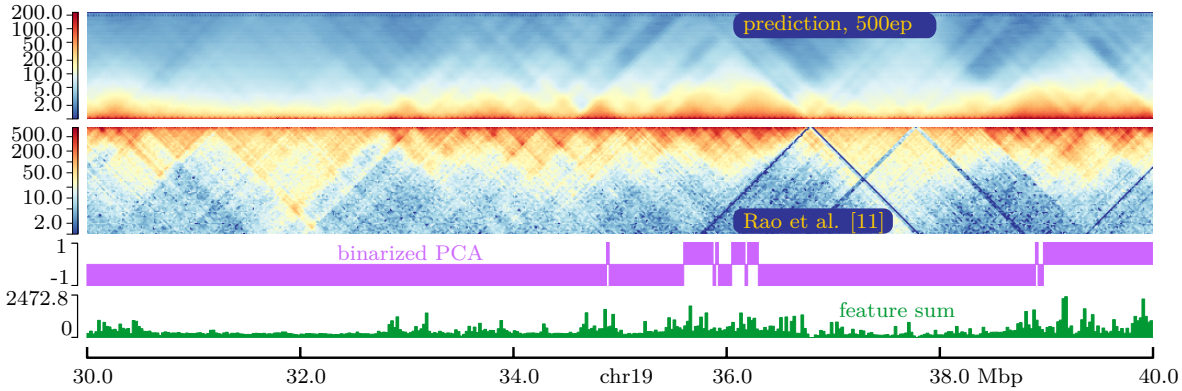
(f) Learning progress

Figure 33: Results / metrics, DNN with combined loss function (MSE, TV, VGG-16), test chromosomes



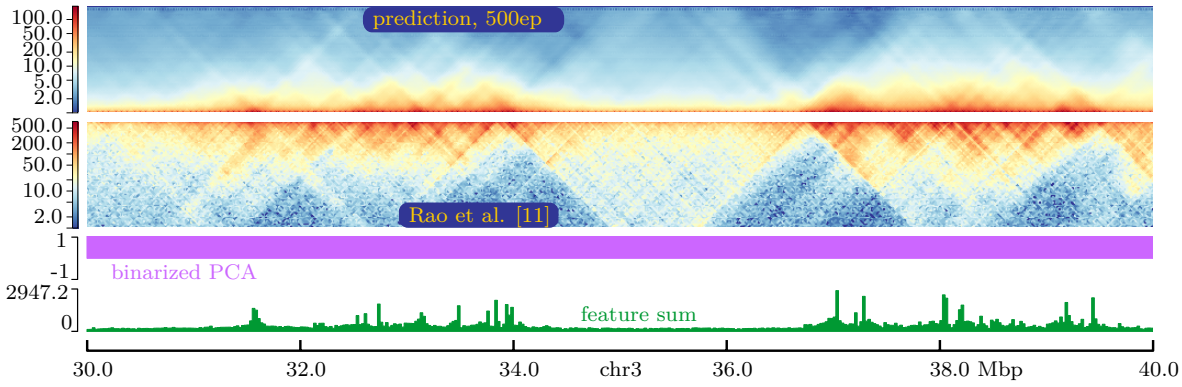
(a) Example region 1

see section 4.6 for track explanations



(b) Example region 2

see section 4.6 for track explanations



(c) Example region 3

Figure 34: Example predictions GM12878 → K562, DNN with combined loss function (MSE, TV, VGG-16), 500 epochs

Long training times forbade a targeted parameter tuning by grid- or tree-search, so the results presented in this section should not be interpreted as the optimal ones achievable by a score-based loss function.

5.1.5 Results for different binsizes and windowsizes

To assess predictions at larger bin sizes, five different approaches were compared, cf. section 3.1.5:

a) "50k direct":
directly training a network at bin size 50 kbp and predicting at that same bin size

b) "initial 25k coarsened":
coarsening the results of the initial network discussed above (section 5.1.1) by summarizing bins via `cooler coarsen`, cf. section 4.1

c) "initial 25k→50k":
using the initial network trained at 25 kbp (cf. section 5.1.1) to predict at 50 kbp

d) "25k+50k→50k":
predicting at 50 kbp from a network *simultaneously* trained with bin sizes 25 and 50 kbp

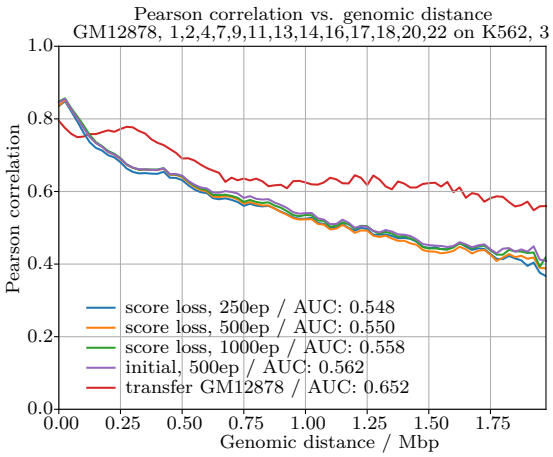
e) "25k+50k→25k":
predicting at 25 kbp from a network *simultaneously* trained with bin sizes 25 and 50 kbp

The best Pearson correlations at bin size 50 kbp were generally obtained either by coarsening the initial results to 50 kbp (method b) or by taking the network trained at 25 kbp for predicting at 50 kbp (method c), fig. 37. Compared to coarsening, the latter approach had the advantage of doubling the window size (in base pairs) and it worked better for test chromosome 21.

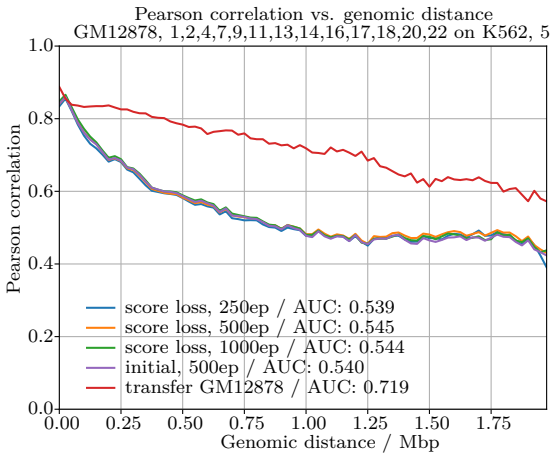
Looking at the corresponding matrix plots, the desired effect of making larger structures more prominent by increasing the bin size was only partially achieved, fig. 39. While all larger structures in the example cutout of test chromosome 3 indeed looked more prominent, fig. 39c, no obvious improvement was observed for the medium-sized structures in the example regions of chromosome 19 and 21, figures 39a and 39b.

Direct predictions at bin size 50 kbp (method a) were definitely not better than indirect methods derived from networks trained at 25 kbp. Both the Pearson correlations and the matrix plots were clearly better for method b) and c), figures 37, 38 and 39. It is not known why the direct predictions turned out worse. Potential reasons include the reduced number of samples (cf. table 3, p. 29) and the binning process, or a combination of both. However, first investigations showed that binning the proteins using the maximum instead of the mean across the 50 kbp-bins, cf. section 4.2, did not improve the results.

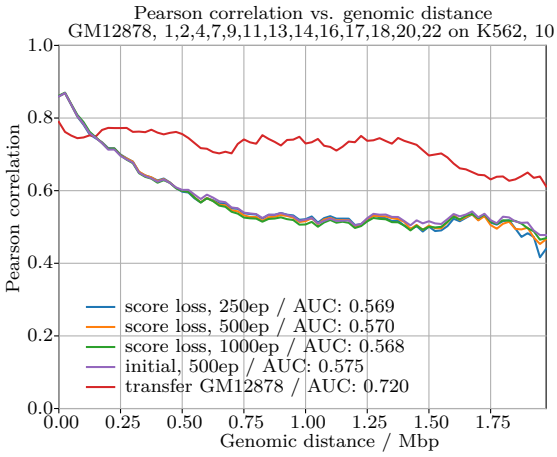
Notably, the training process for the direct prediction at bin size 50 kbp (method a) diverged after about 420 epochs. One possible reason for this could be too high a learning rate, which could have been avoided by decreasing the learning rate over time. However, no further investigations were made into the case, because the divergence occurred only after overfitting, fig. 37f, and was thus not seen as too problematic here. The minimum validation error was reached after about 150 epochs and thus about 100 epochs earlier than in the initial setup at 25 kbp. This is



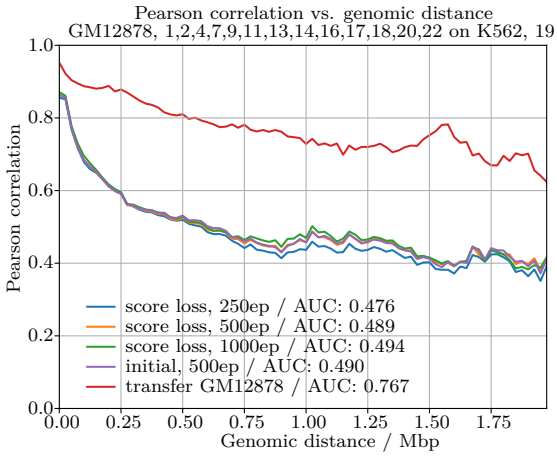
(a) chr3



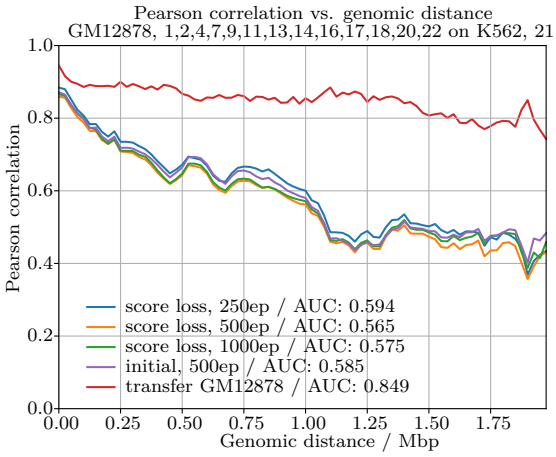
(b) chr5



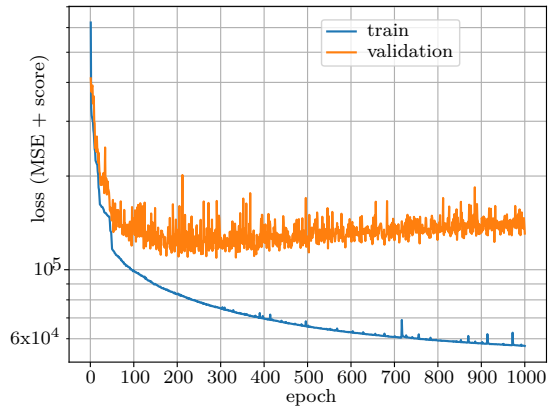
(c) chr10



(d) chr19

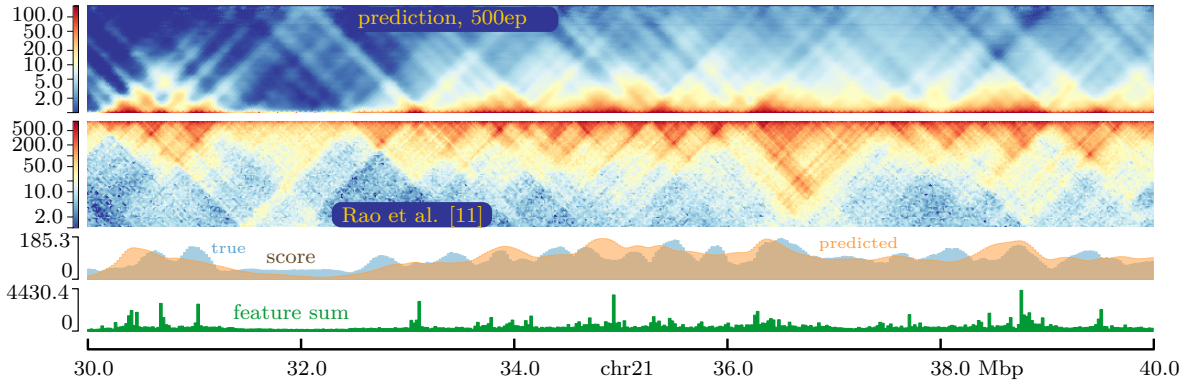


(e) chr21



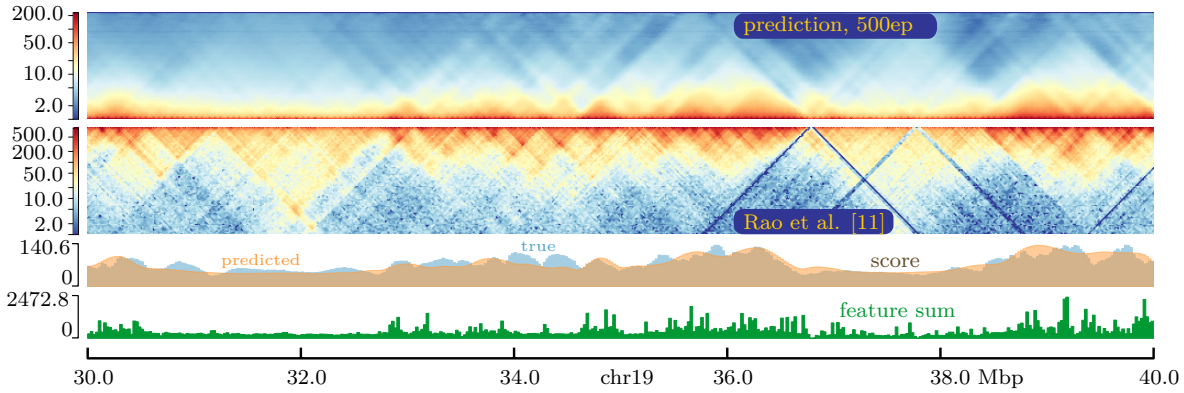
(f) Learning progress

Figure 35: Results / metrics, DNN with score-based loss function, test chromosomes ($\lambda_{MSE} = 1.0,\ \lambda_{score} = 100,\ ds = 12$)



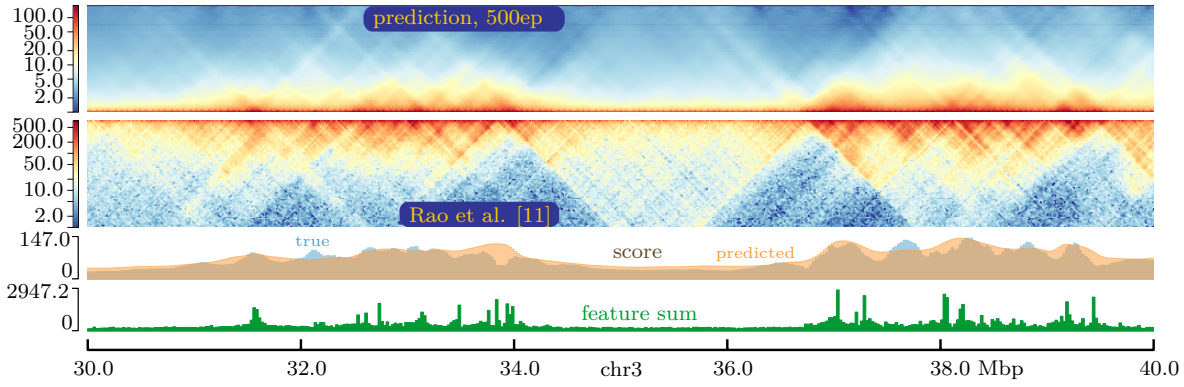
(a) Example region 1

see section 4.6 for track explanations



(b) Example region 2

see section 4.6 for track explanations

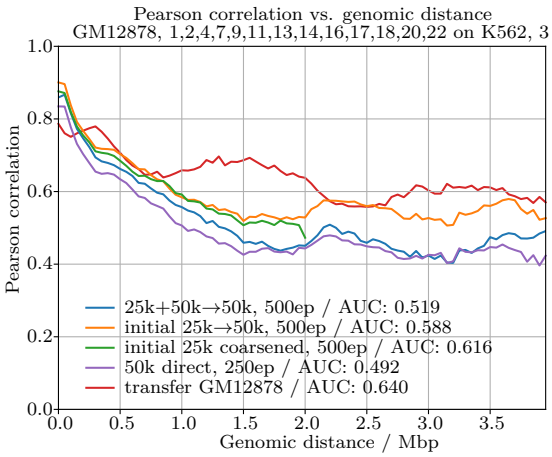


(c) Example region 3

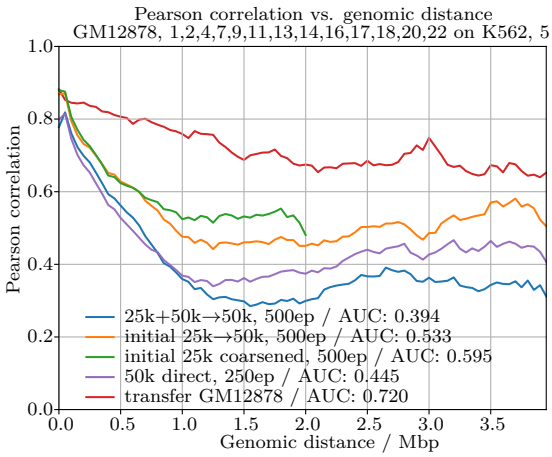
Figure 36: Example predictions GM12878 → K562, DNN with score-based loss function, 500 epochs

not surprising, since there are only about half as many training samples at 50 kbp compared to 25 kbp, cf. table 3 (p. 29).

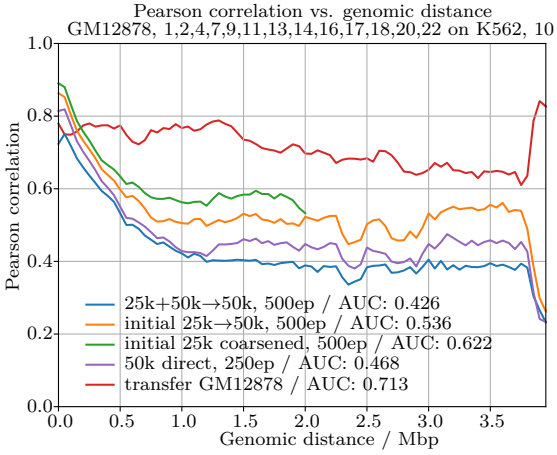
Simultaneously training a network with matrix- and feature bin sizes of 25 kbp and 50 kbp (methods d, e) turned out unproblematic with regard to convergence, fig. 40f, but the Pearson correlations when predicting at both 25 kbp and 50 kbp were – often significantly – worse than the initial predictions at the respective bin size, fig. 37 ("25k+50k→50k") and fig. 40 ("25k+50k→25k"). Looking into the matrix plots shown in fig. 41, all predictions seemed equally useless and definitely worse than the results obtained by the other approaches investigated thus far. It could not be clarified what caused the improvement in Pearson correlations for chromosome 21 compared to the initial predictions at 25 kbp, fig. 40e, but it is interesting that even predictions with such a high degree of blurriness as in fig. 41a can reach an AUC of around 0.65.



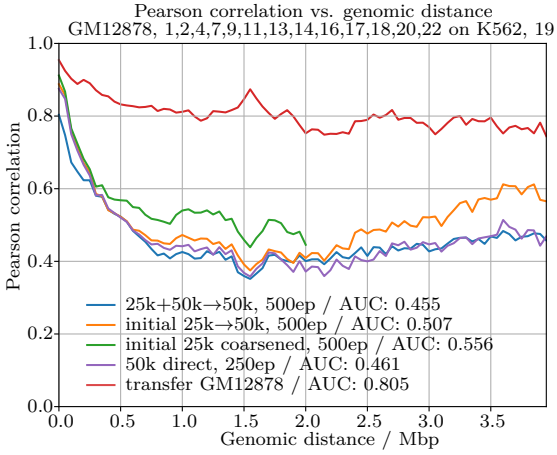
(a) chr3



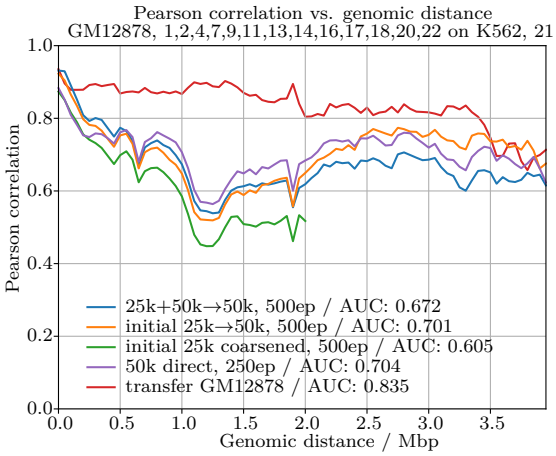
(b) chr5



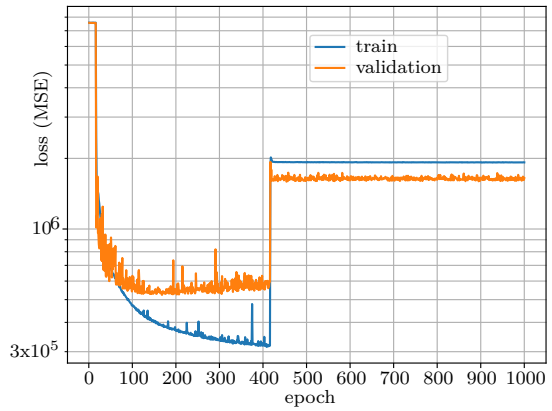
(c) chr10



(d) chr19

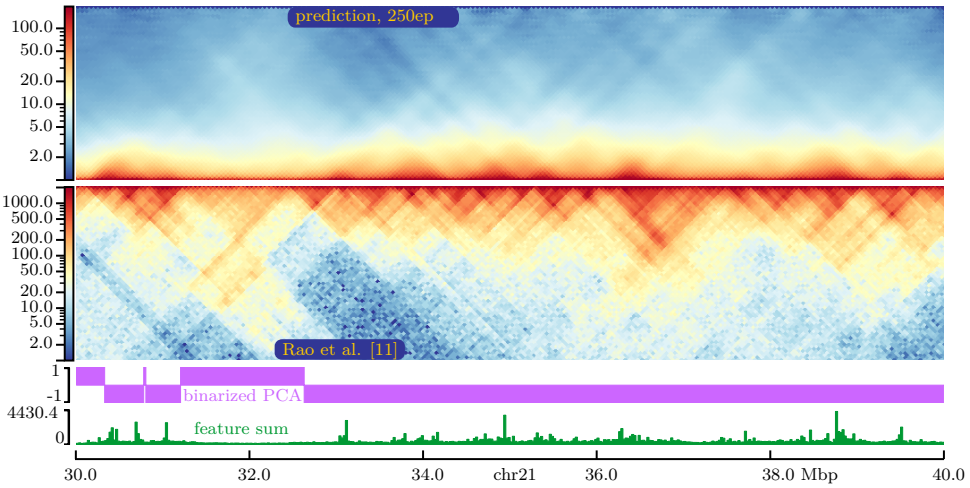


(e) chr21



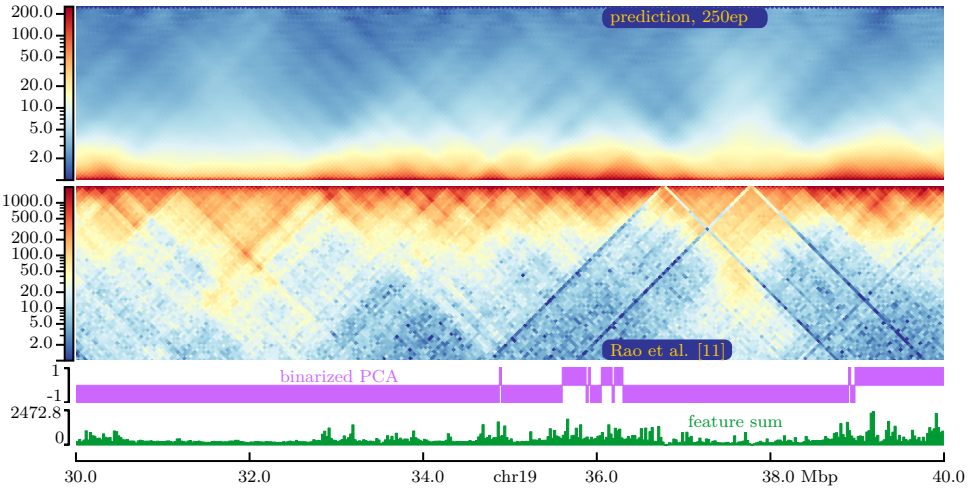
(f) Learning progress 50k direct

Figure 37: Results / metrics, various DNNs at 50 kbp



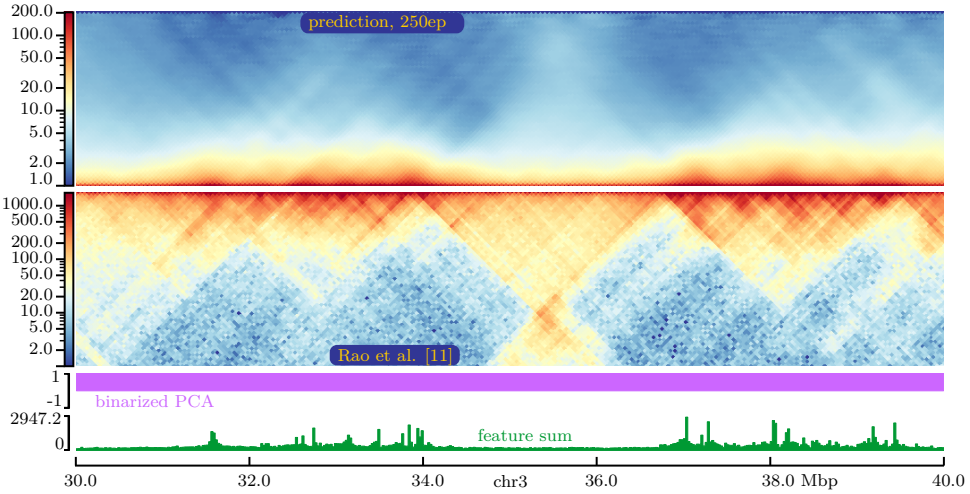
(a) Example region 1

see section 4.6 for track explanations



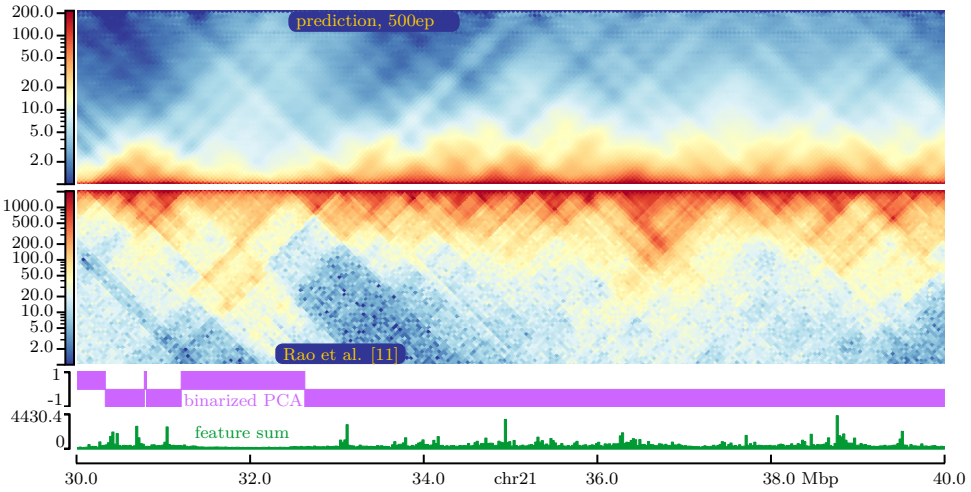
(b) Example region 2

see section 4.6 for track explanations



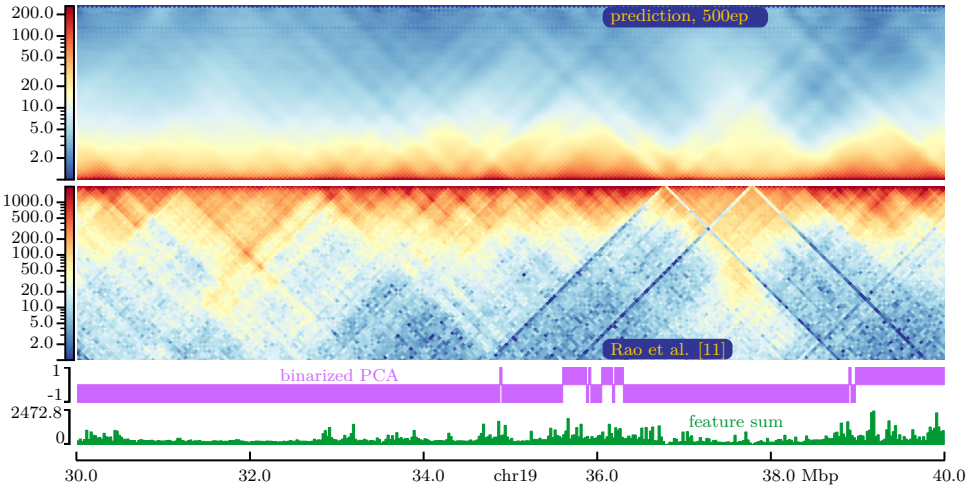
(c) Example region 3

Figure 38: Example predictions GM12878 → K562, DNN at 50 kbp direct, 250 epochs



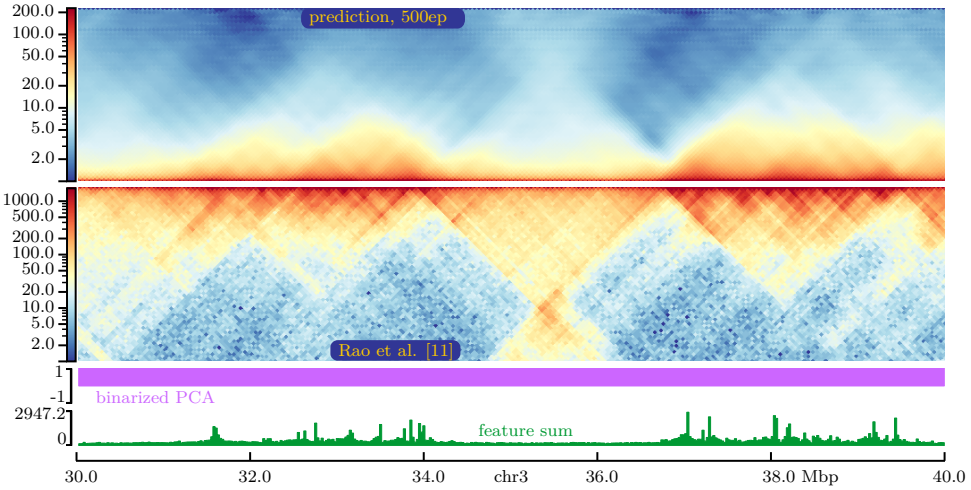
(a) Example region 1

see section 4.6 for track explanations



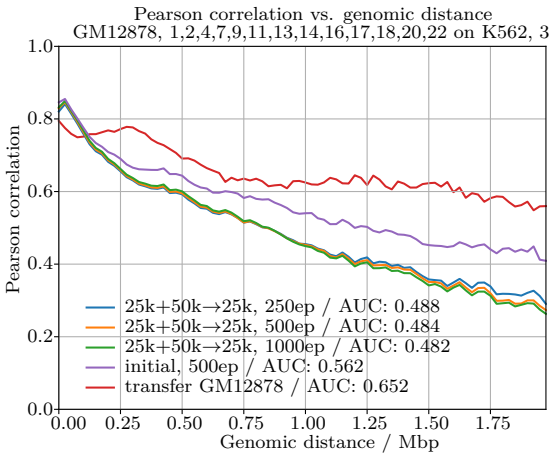
(b) Example region 2

see section 4.6 for track explanations

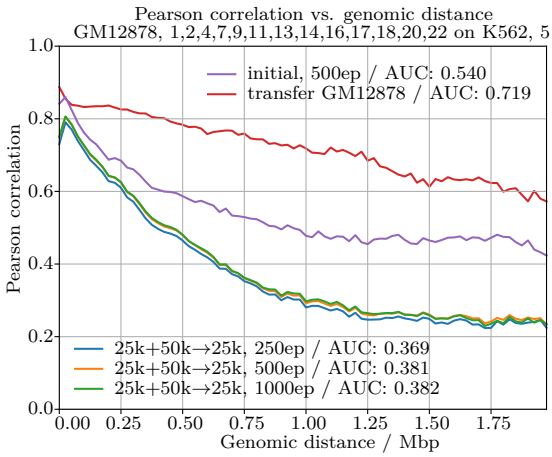


(c) Example region 3

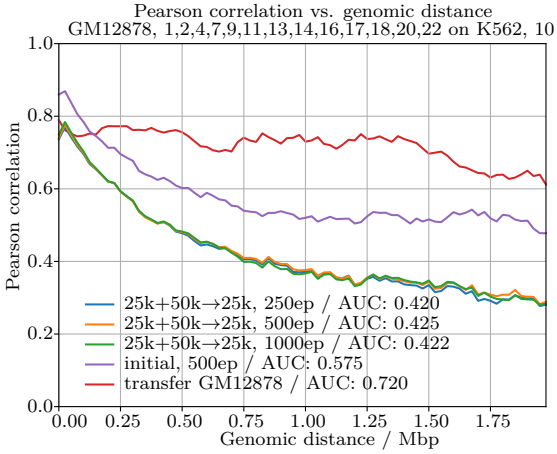
Figure 39: Example predictions GM12878 → K562, DNN trained at 25 kbp predicting at 50 kbp, 500 epochs



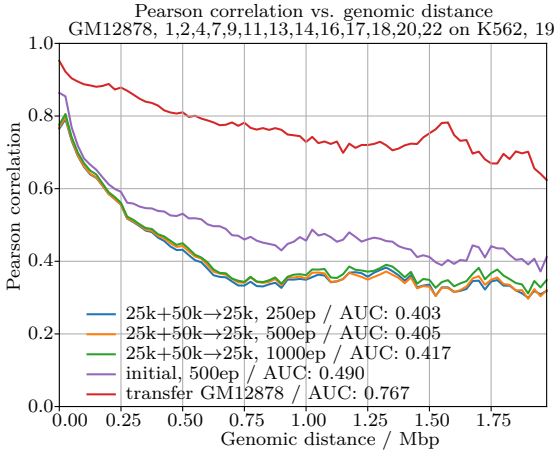
(a) chr3



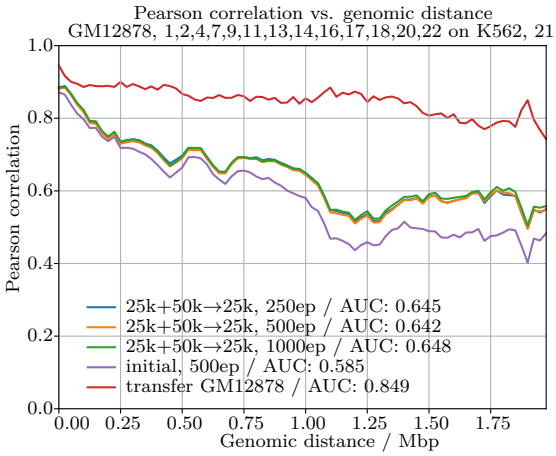
(b) chr5



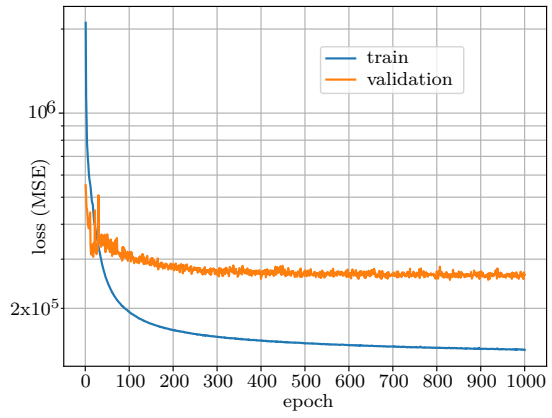
(c) chr10



(d) chr19

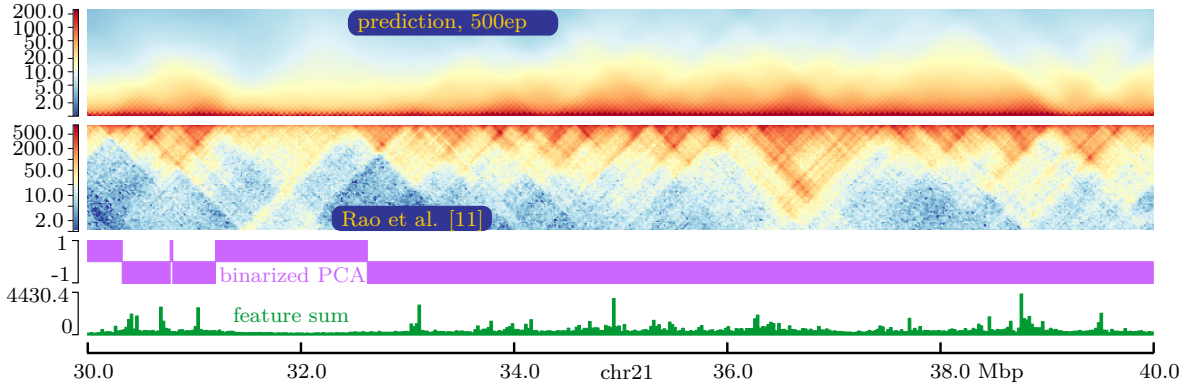


(e) chr21



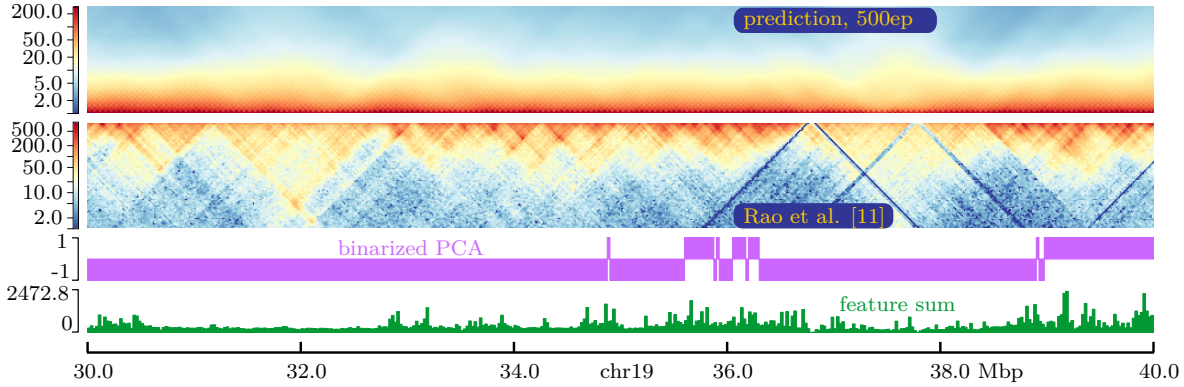
(f) Learning progress

Figure 40: Results / metrics, DNN trained at 25 kbp and 50 kbp simultaneously



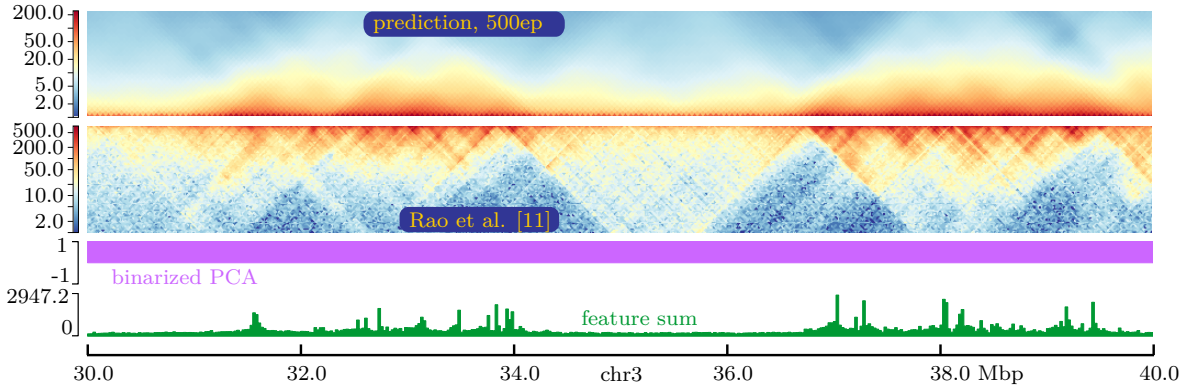
(a) Example region 1

see section 4.6 for track explanations



(b) Example region 2

see section 4.6 for track explanations



(c) Example region 3

Figure 41: Example predictions GM12878 → K562, DNN trained at 25 kbp and 50 kbp simultaneously, 25 kbp, 500 epochs

5.2 Hi-cGAN approaches

In the following three subsections, the results from the conditional generative adversarial networks will be presented. Although there is still room for improvements, at least one of the three cGAN-variants under investigation showed good performance.

5.2.1 cGAN with DNN embedding

The cGAN with DNN-embedding showed interesting results. All in all, the training process was smooth and converged after around 60 epochs, mostly using the parameters suggested by Isola et al. [49] (cf. 4.8.1). Although *pix2pix* has shown fast convergence in other applications, it was surprising that this still held after the changes made to the original network, especially adding the embedding network. While the Pearson correlations were mostly worse than the ones of the DNN presented above, compare figures 61 and 42, the matrices mostly looked visually better, showing more distinct boundaries between interacting and non-interacting regions, compare figures 62 and 43.

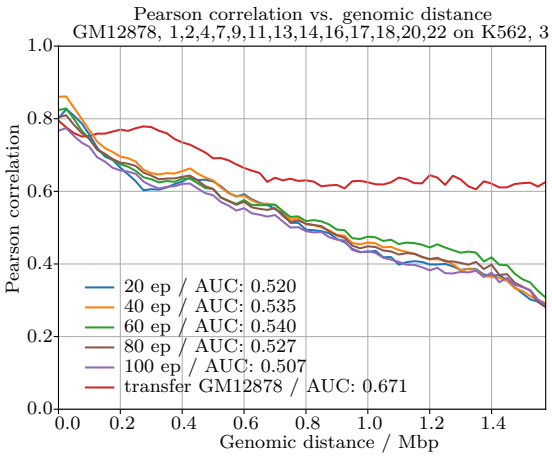
Using weight transfer from a pre-trained DNN for the embedding networks further stabilized the training process and made the discriminator reach a stable value of around 0.693 ($\approx -\log 0.5$) on epoch average after only 2 epochs, while the generator validation loss reached a stable minimum value after about 15 epochs, fig. 44f. However, the resulting predictions did not really improve. While the Pearson correlations showed showed slightly better values for the pre-trained DNN-embedding compared to the non-pre-trained, fig. 44, the matrices were visually clearly worse than without pre-training, and also worse than the results from the DNN alone, fig. 45.

5.2.2 cGAN with CNN embedding

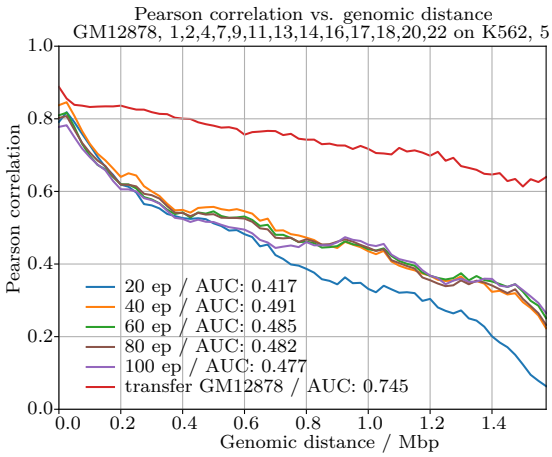
The DNN embedding also allowed for a stable training of the cGAN, whereby the discriminator loss reached stable epoch-average values after 20 to 30 epochs, depending on windowsize, fig. 46f. For windowsize $w = 64$, the Pearson correlations showed worse values than the DNN-approach or the cGAN with DNN embedding, fig. 46. Nevertheless, the predicted matrices looked visually better, at least for the given test cutouts shown in section 5.2.2. For windowsizes $w \in \{128, 256\}$, the Pearson correlations often reached and partially exceeded the baseline ("transfer GM12878 to K562"), and the matrix plots were visually in good accordance with the targets, figures 48 to 51.

Predictions with swapped training- and test cell line, i. e. with data from K562 for training and data from GM12878 for prediction were investigated only for windowsize $w = 256$, which seemed most promising. Even though the training process was not as stable as for other investigations at smaller windowsizes, the predictions indeed showed Pearson correlations largely above the baseline ("transfer K562 to GM12878") and matrix plots were in good accordance with their respective targets, cf. fig. 63 and 64.

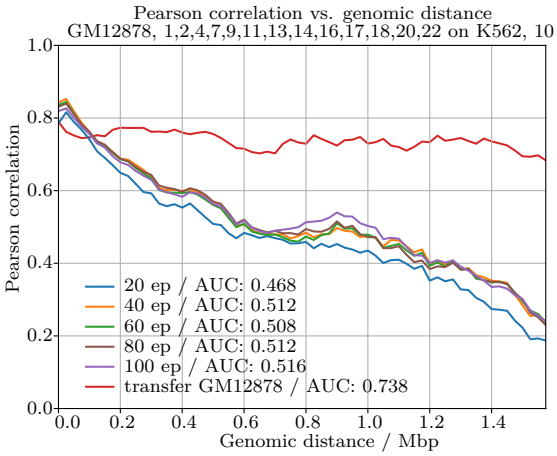
Average training times per epoch were around 110 min for windowsize $w = 256$ and 28 min and 14 min for windowsizes $w = 128$ and $w = 64$, respectively. Here, training was performed on machine 1 for windowsizes $w = \{64, 128\}$ and on machine 2 for windowsize $w = 256$, due to memory limitations on machine 1, cf. section 7.3.



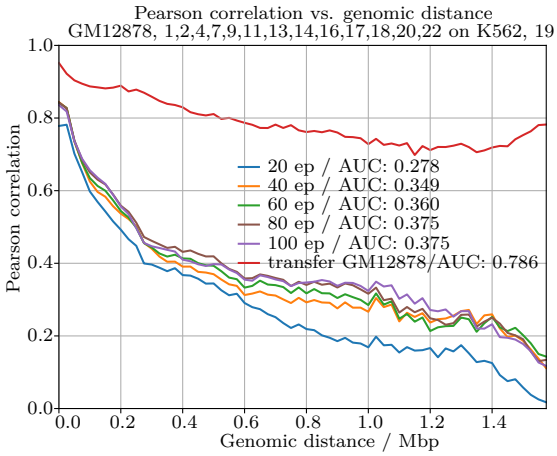
(a) chr3



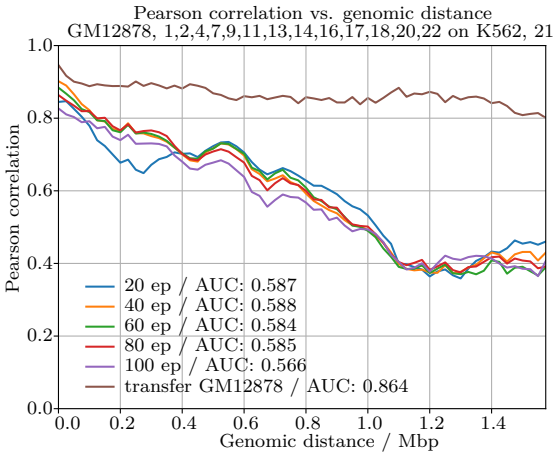
(b) chr5



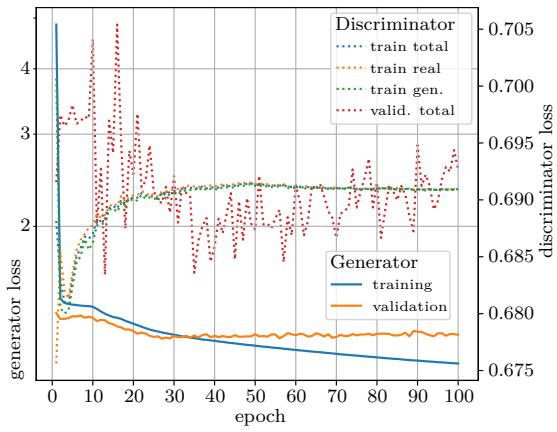
(c) chr10



(d) chr19

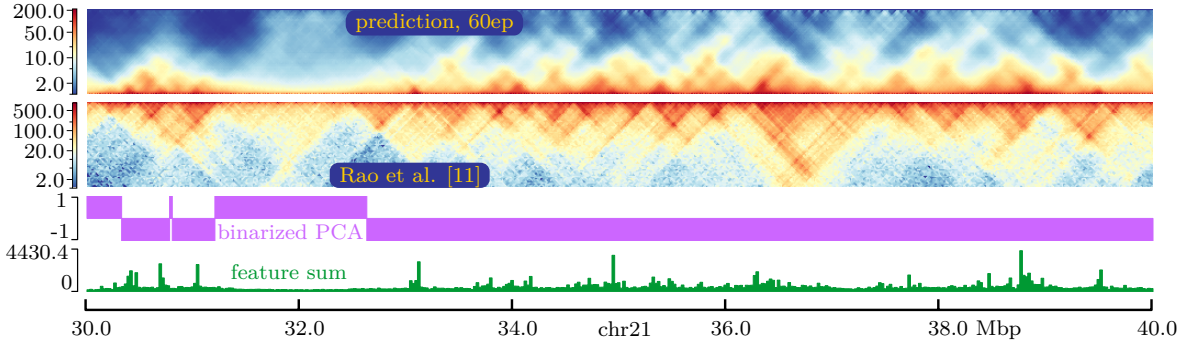


(e) chr21



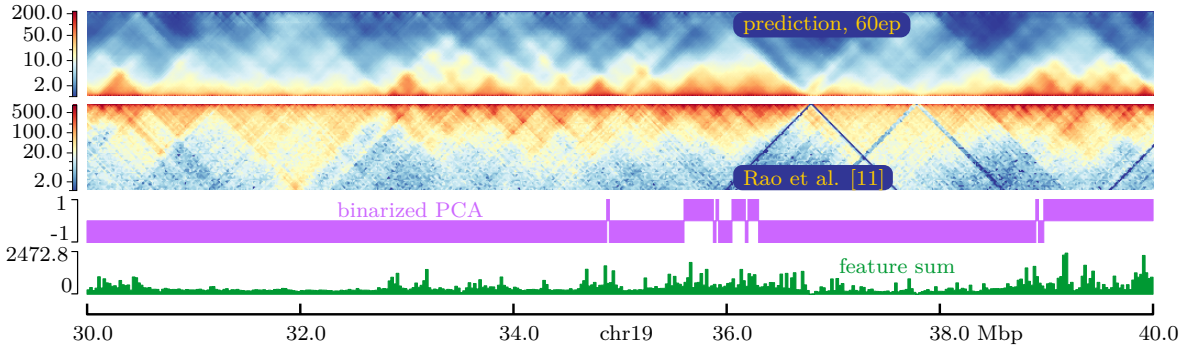
(f) Learning progress

Figure 42: Results / metrics cGAN, DNN embedding, no pre-training, $w = 64$, test chromosomes



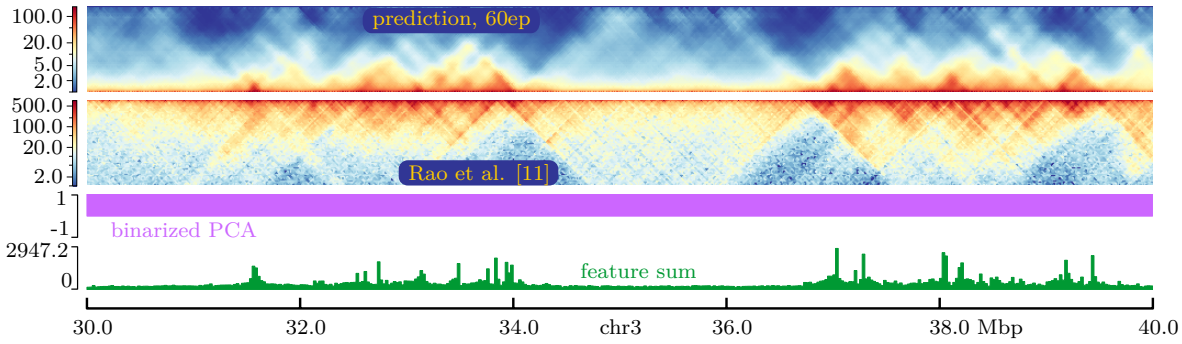
(a) Example region 1

see section 4.6 for track explanations



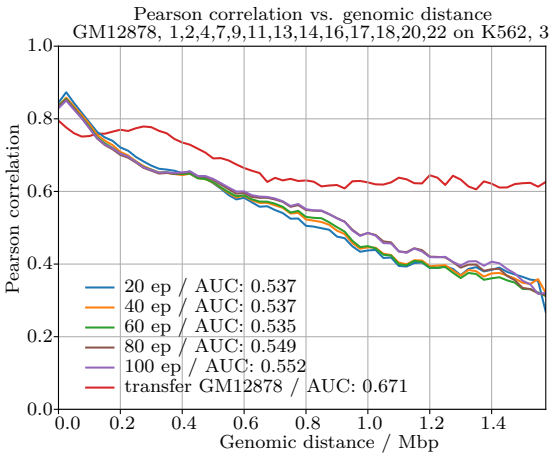
(b) Example region 2

see section 4.6 for track explanations

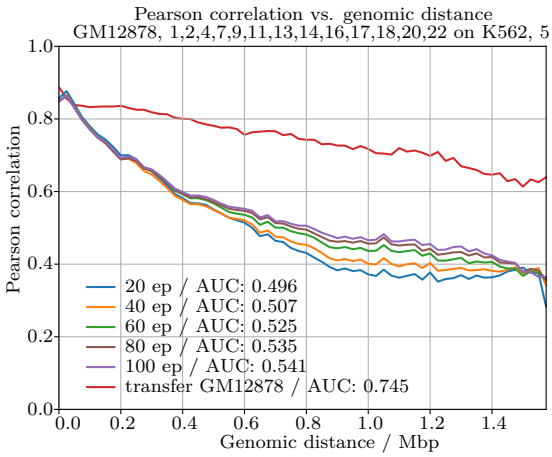


(c) Example region 3

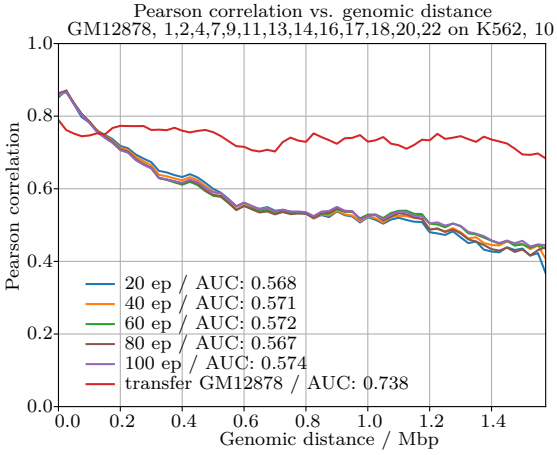
Figure 43: Example predictions GM12878 → K562, cGAN, DNN embedding, no pre-training, $w = 64$, 60 epochs



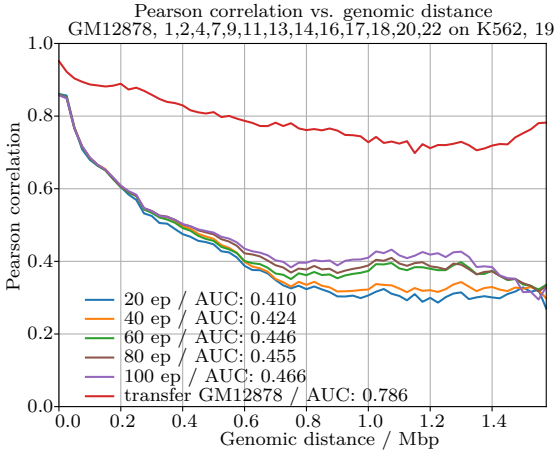
(a) chr3



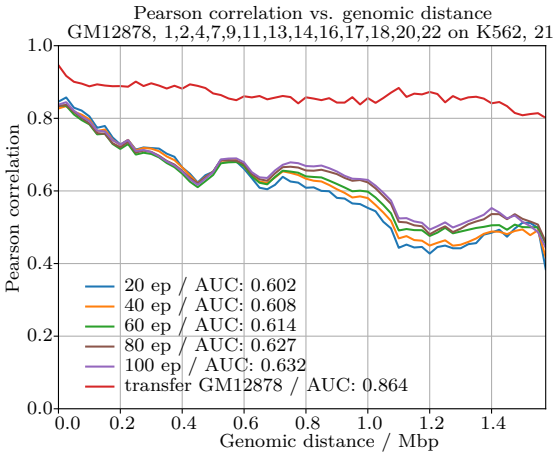
(b) chr5



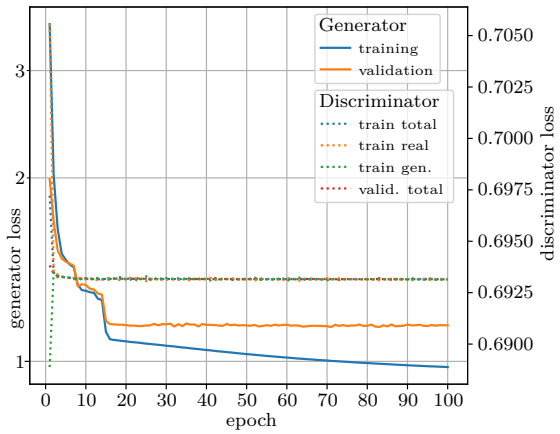
(c) chr10



(d) chr19

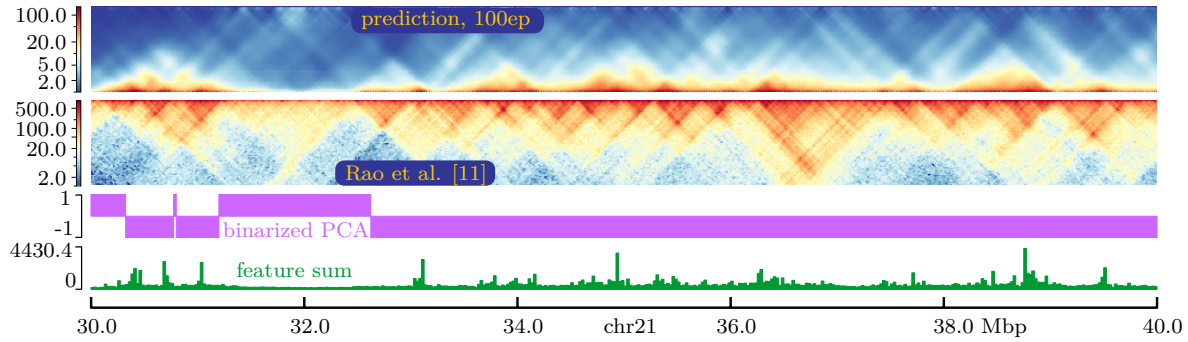


(e) chr21



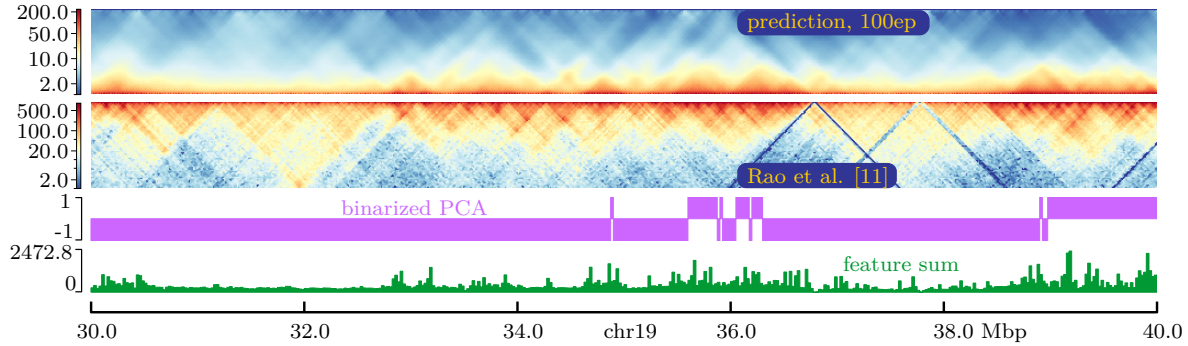
(f) Learning progress

Figure 44: Results / metrics, cGAN, DNN embedding, pre-trained, $w = 64$, test chromosomes



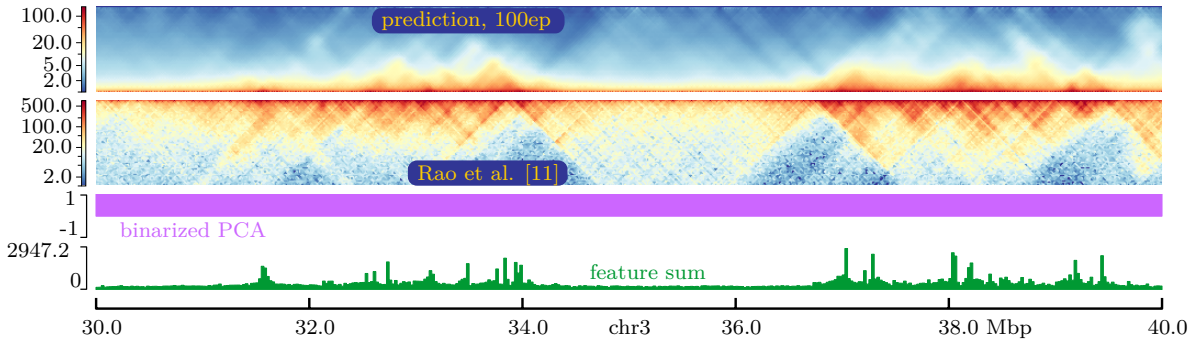
(a) Example region 1

see section 4.6 for track explanations



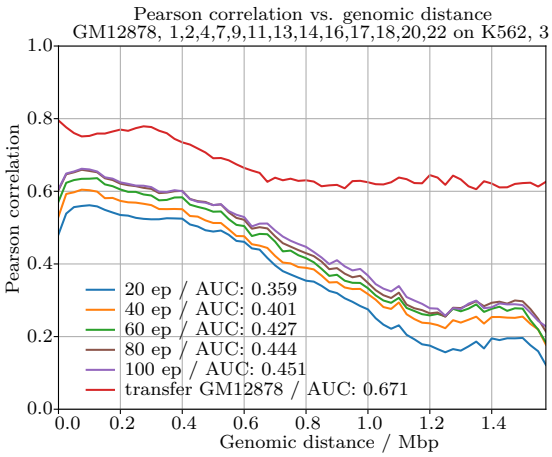
(b) Example region 2

see section 4.6 for track explanations

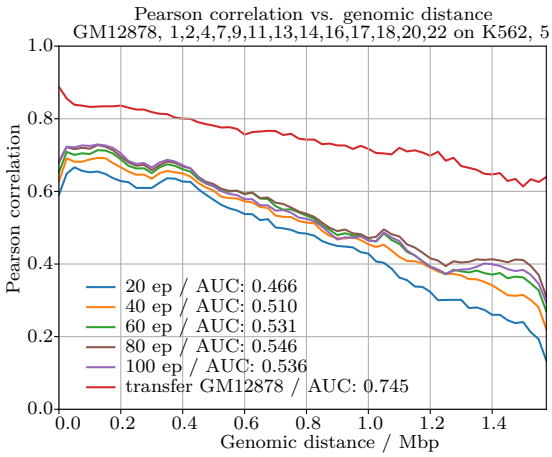


(c) Example region 3

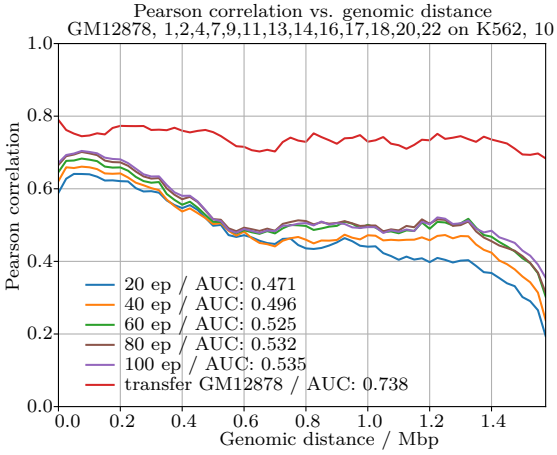
Figure 45: Example predictions GM12878 → K562, cGAN, DNN embedding, pre-trained, $w = 64$, 100 epochs



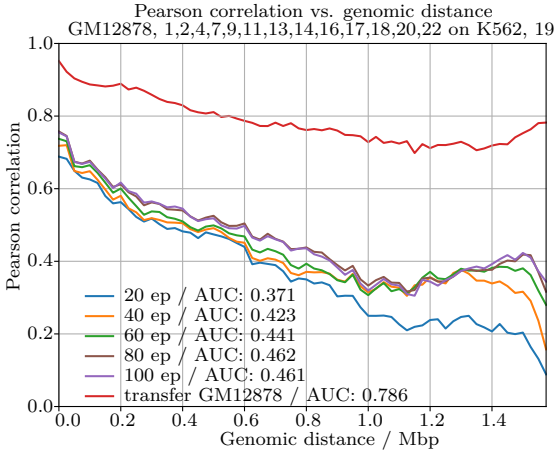
(a) chr3



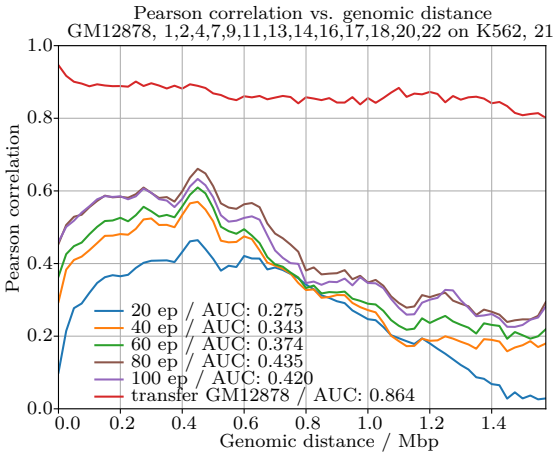
(b) chr5



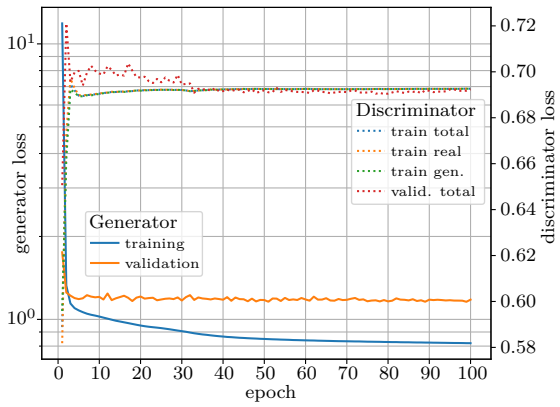
(c) chr10



(d) chr19

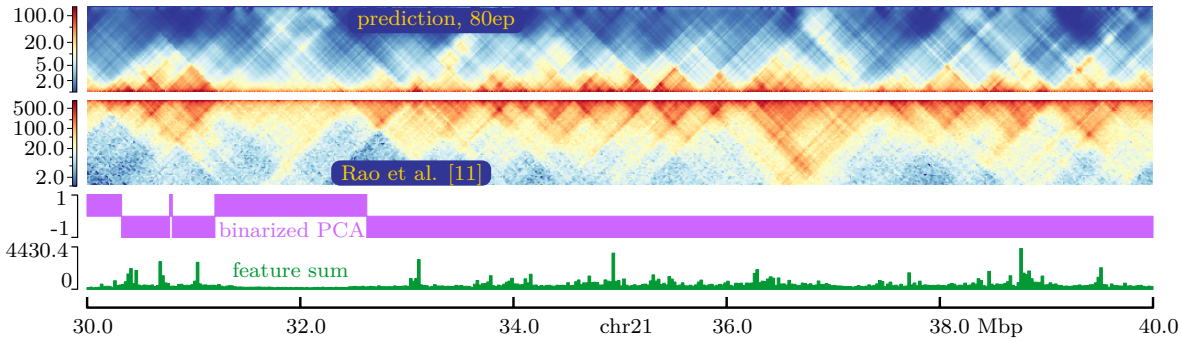


(e) chr21



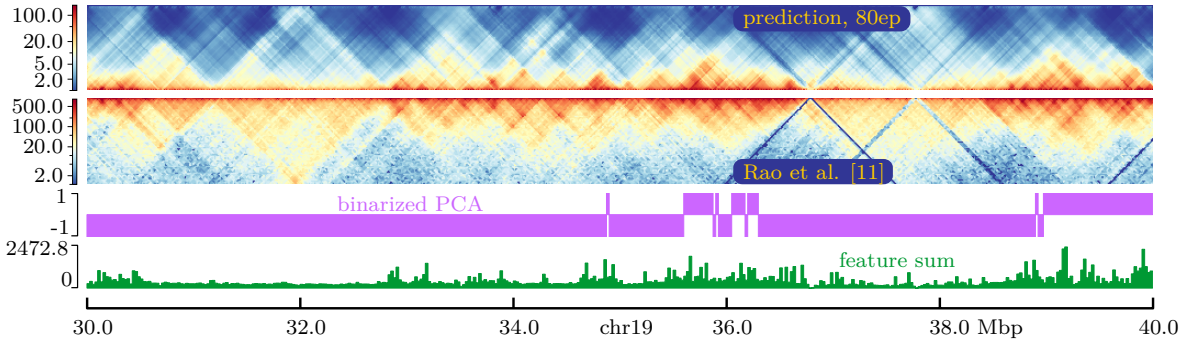
(f) Learning progress

Figure 46: Results / metrics cGAN, CNN embedding, $w = 64$, test chromosomes



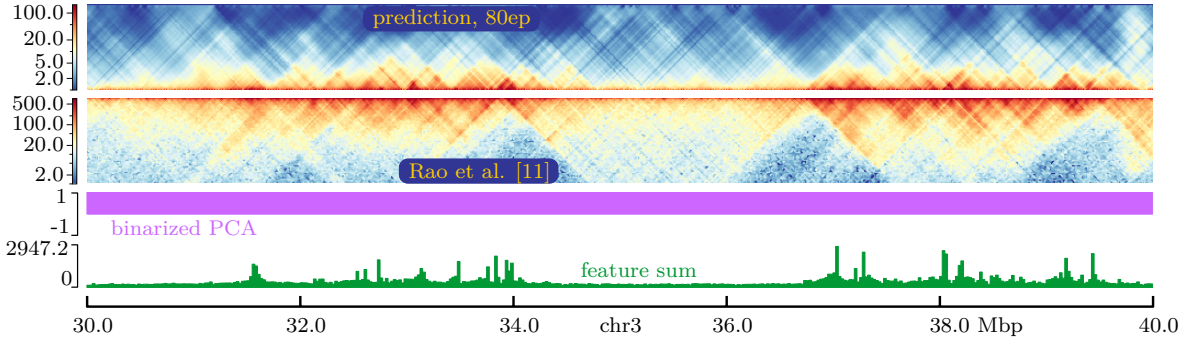
(a) Example region 1

see section 4.6 for track explanations



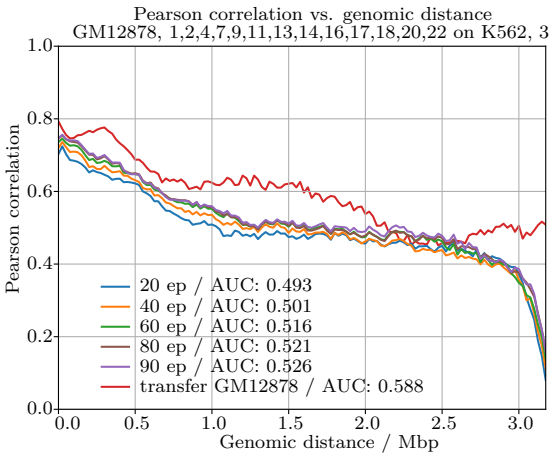
(b) Example region 2

see section 4.6 for track explanations

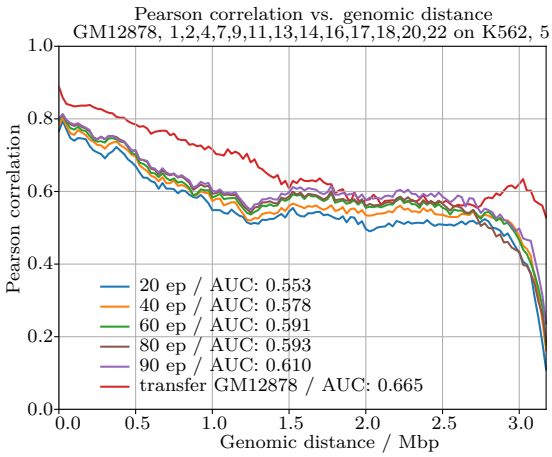


(c) Example region 3

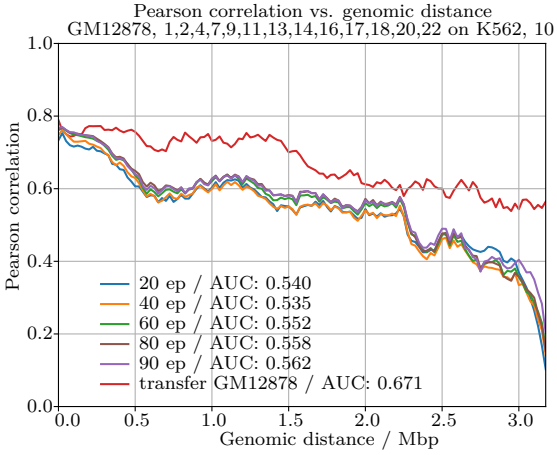
Figure 47: Example predictions GM12878 → K562, cGAN, CNN embedding, $w = 64$, 80 epochs



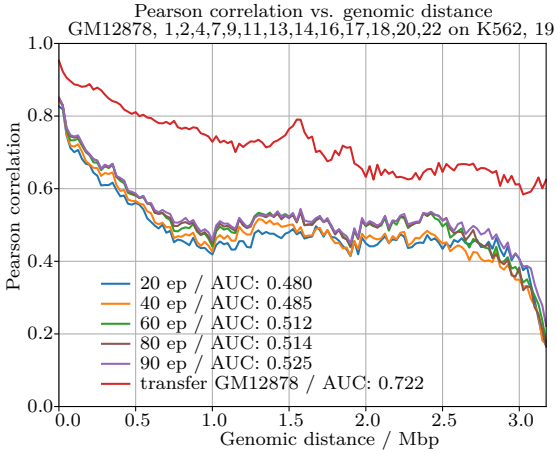
(a) chr3



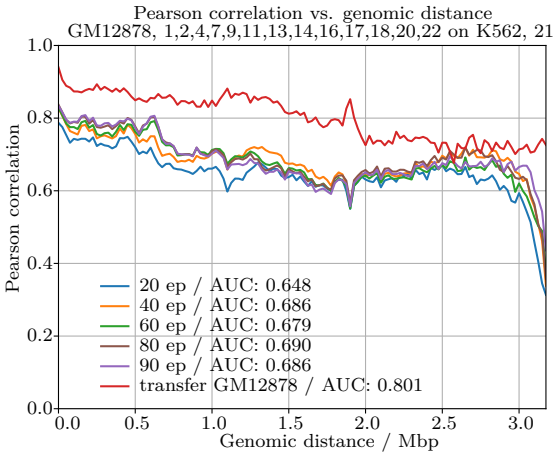
(b) chr5



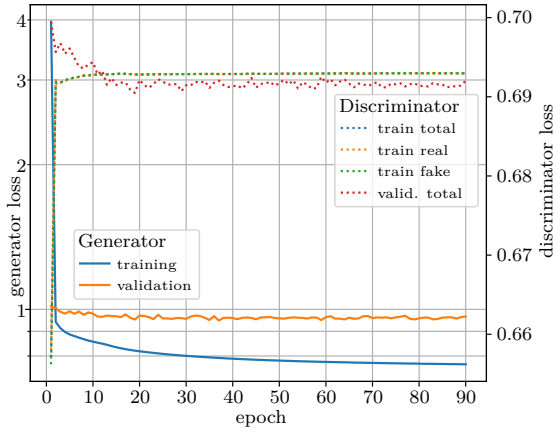
(c) chr10



(d) chr19

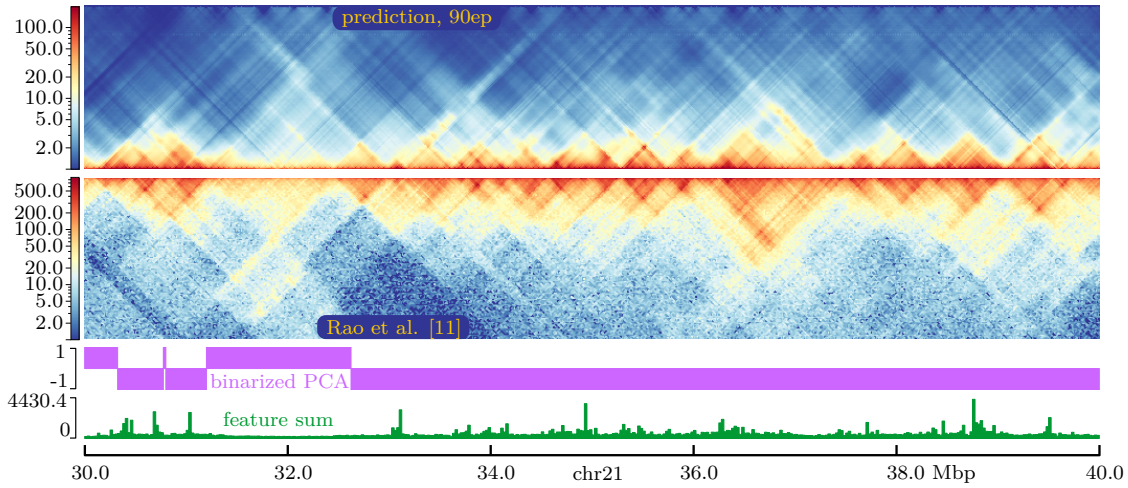


(e) chr21



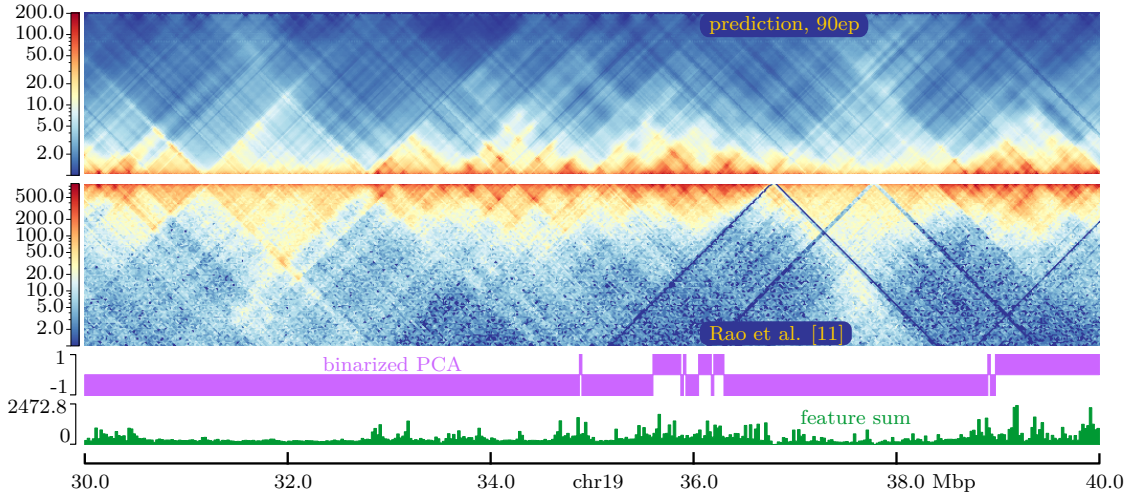
(f) Learning progress

Figure 48: Results / metrics cGAN, CNN embedding. $w = 128$, test chromosomes



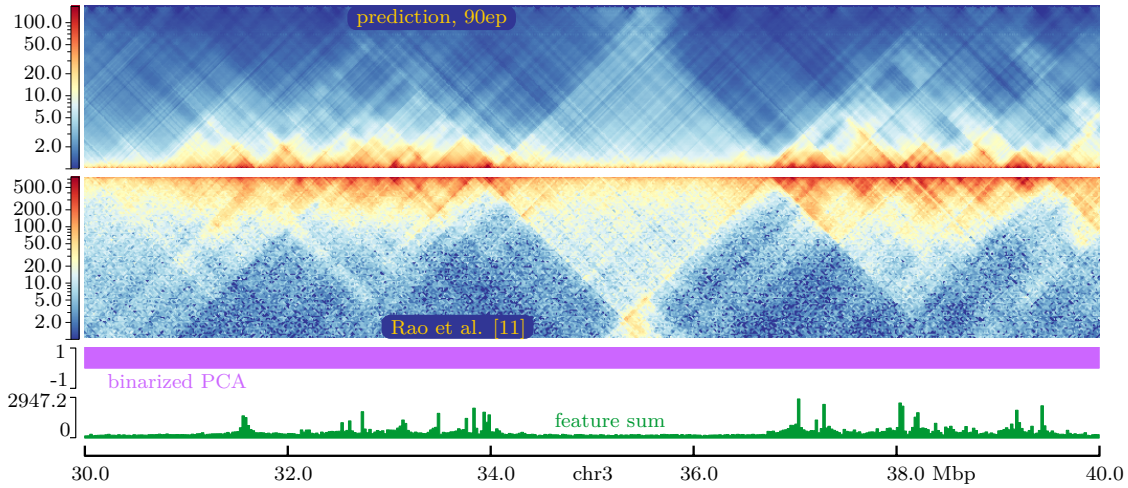
(a) Example region 1

see section 4.6 for track explanations



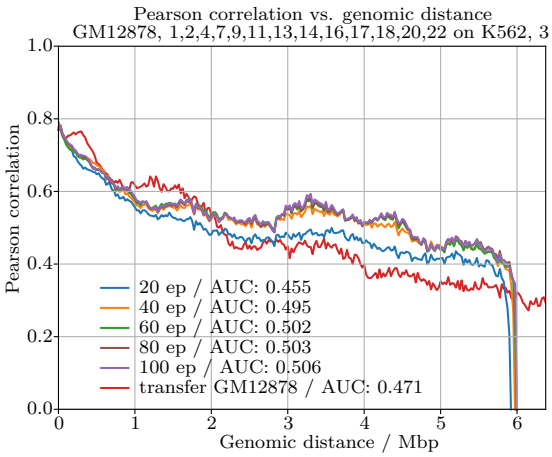
(b) Example region 2

see section 4.6 for track explanations

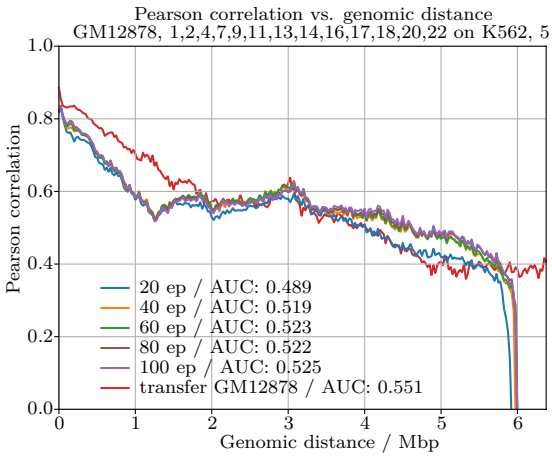


(c) Example region 3

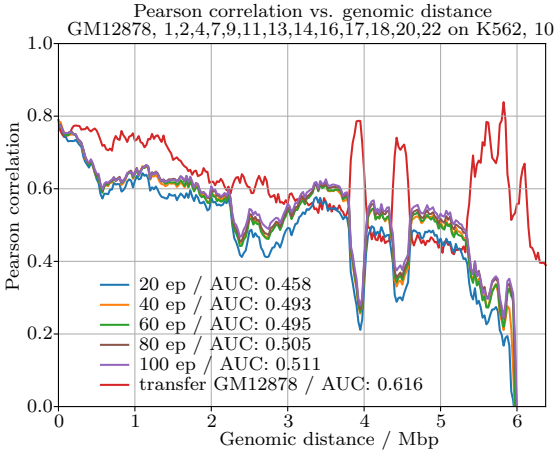
Figure 49: Example predictions GM12878 → K562, cGAN, CNN embedding, $w = 128$, 90 epochs



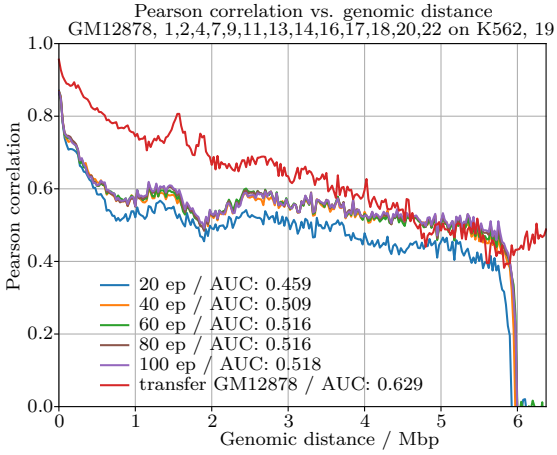
(a) chr3



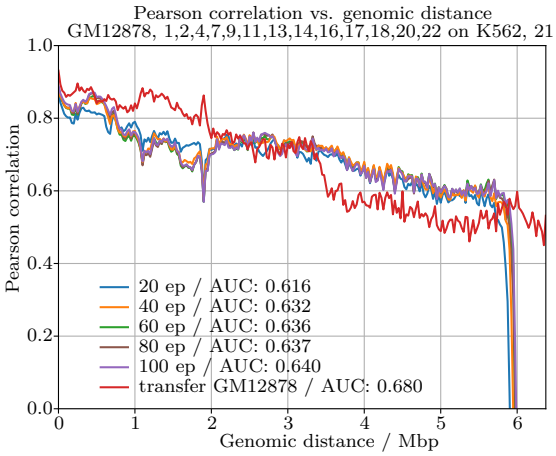
(b) chr5



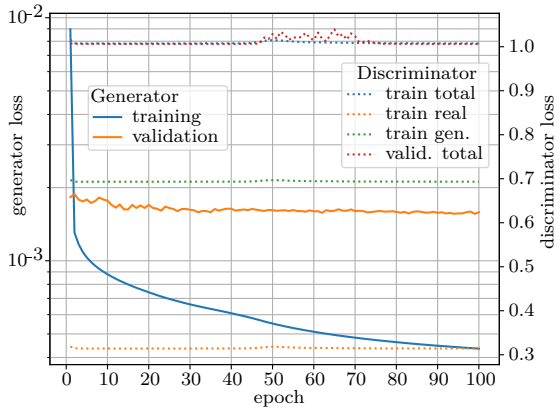
(c) chr10



(d) chr19

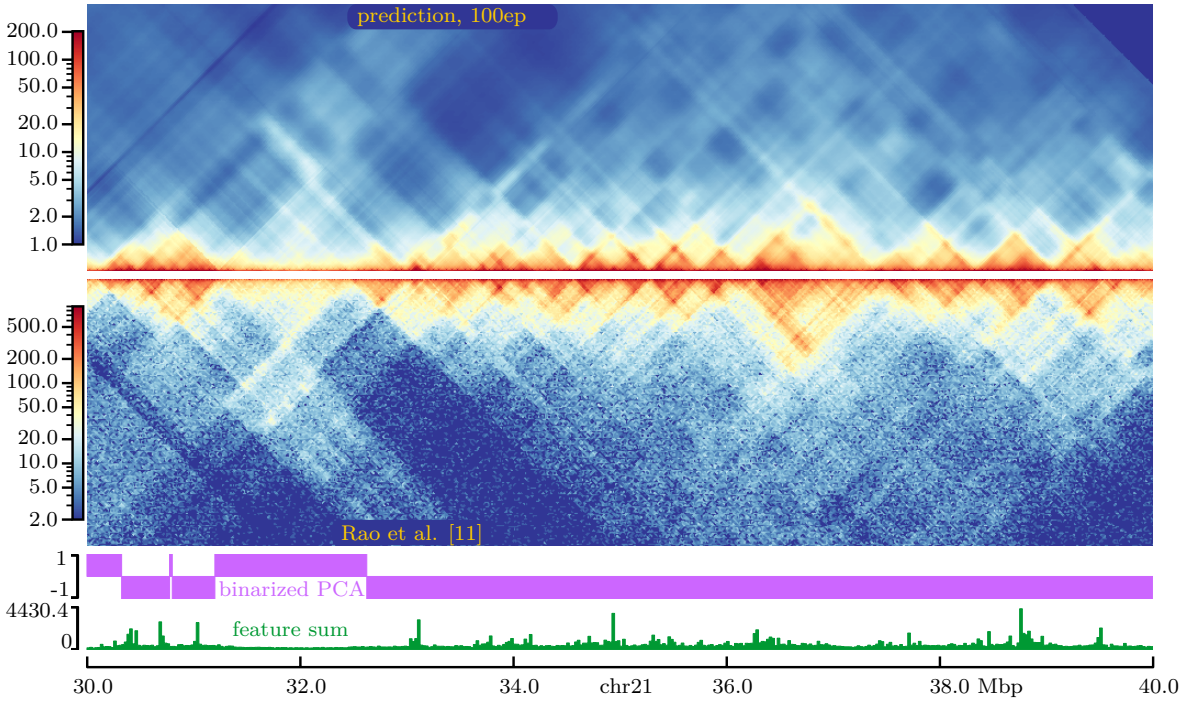


(e) chr21



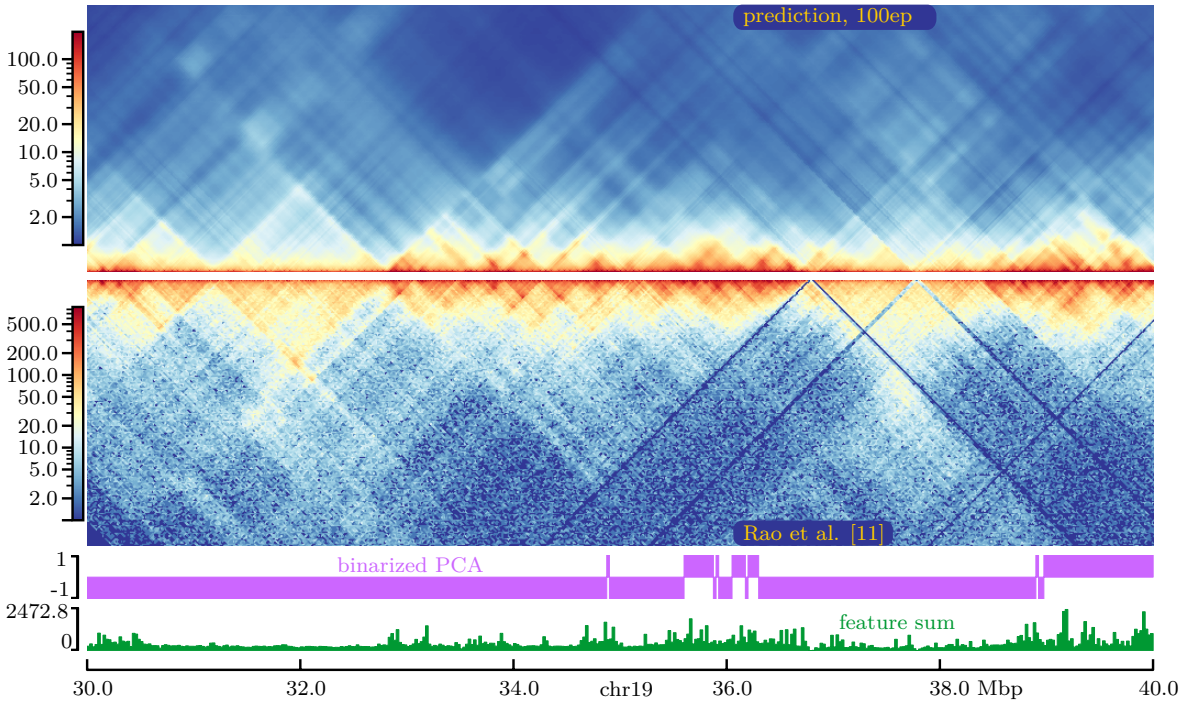
(f) Learning progress

Figure 50: Results / metrics cGAN, CNN embedding, $w = 256$, test chromosomes



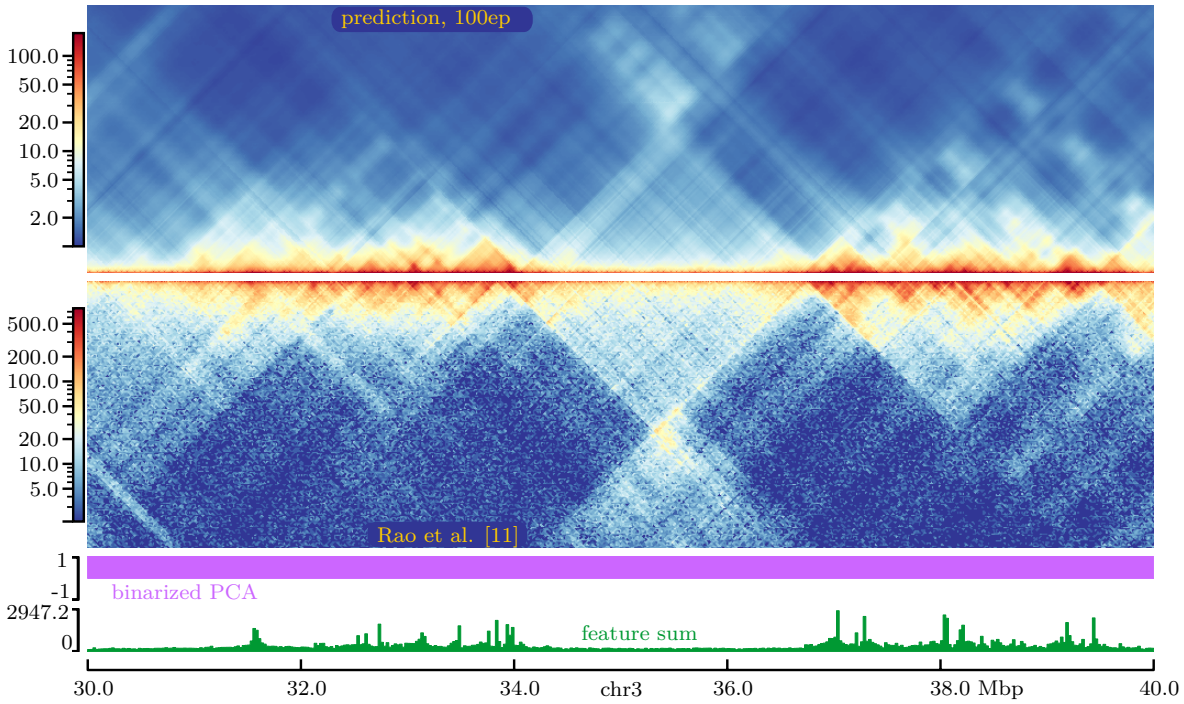
(a) Example region 1

see section 4.6 for track explanations



(b) Example region 2

Figure 51: Example predictions GM12878 → K562, cGAN, CNN embedding, $w = 256$, 100 epochs



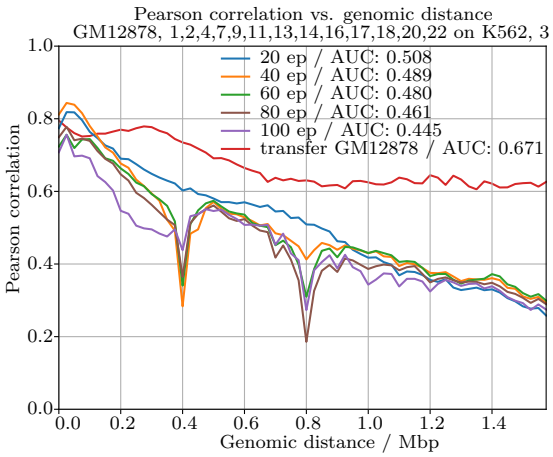
(c) Example region 3

Figure 51: Example predictions GM12878 → K562, cGAN, CNN embedding, $w = 256$, 100 epochs

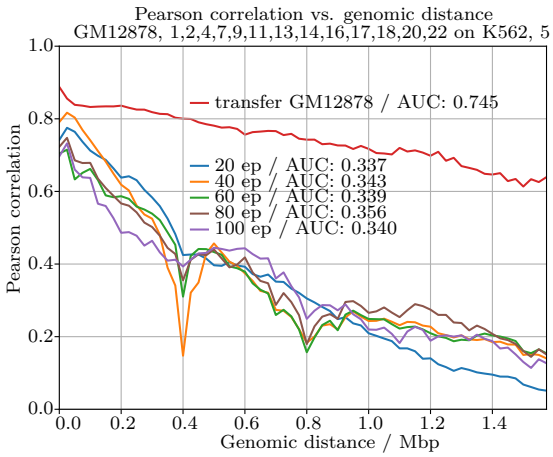
5.2.3 cGAN with mixed DNN / CNN embedding

Using a cGAN with mixed embedding, i. e. DNN-embedding for the generator and CNN-embedding for the discriminator, did not make for a stable training process, 52f. Despite still acceptable Pearson correlations compared to the other approaches shown above, the predicted matrices were useless, fig. 52 and 53. First investigations indicated that the cGAN may have learned to partially ignore the feature input and is thus predicting submatrices that look similar to real Hi-C submatrices, but do not have sufficient correlation with the feature data.

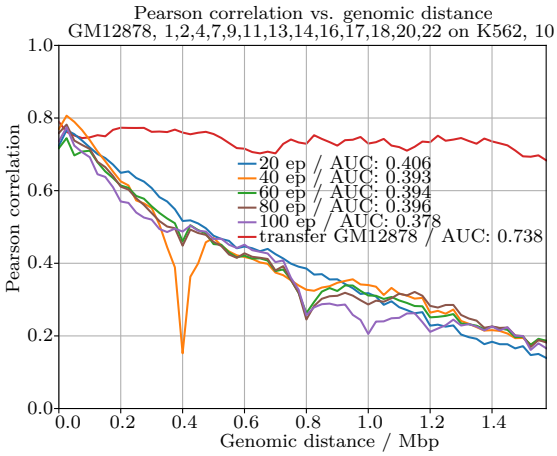
Interestingly, pre-training the DNN improved the situation in this mixed DNN/CNN embedding setup. While the training process still seemed unstable, fig. 54f, the predicted matrices looked much better, similar to the ones from the cGAN with DNN-embedding (non-pre-trained), compare figures 55 and 43. However, the results were still not better than the ones from the CNN embedding alone, cf. section 5.2.2.



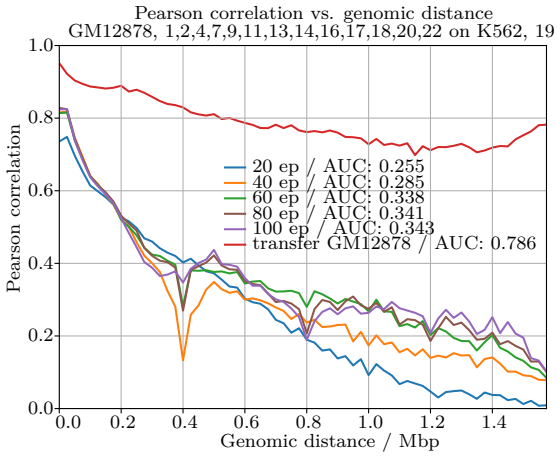
(a) chr3



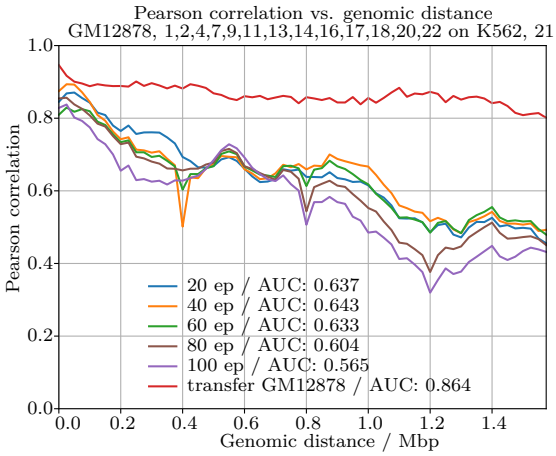
(b) chr5



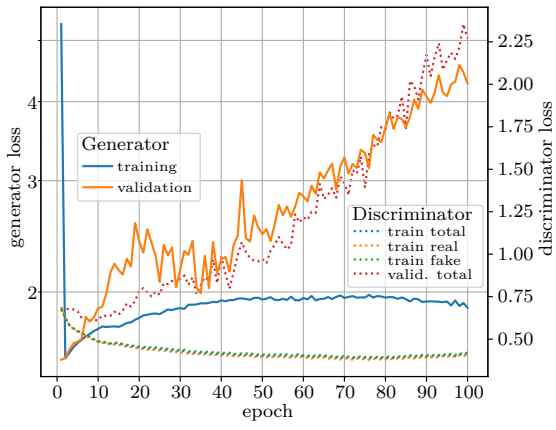
(c) chr10



(d) chr19

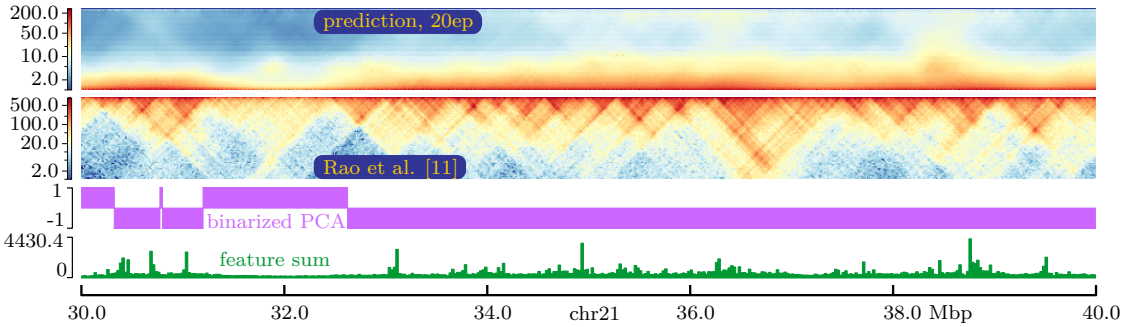


(e) chr21



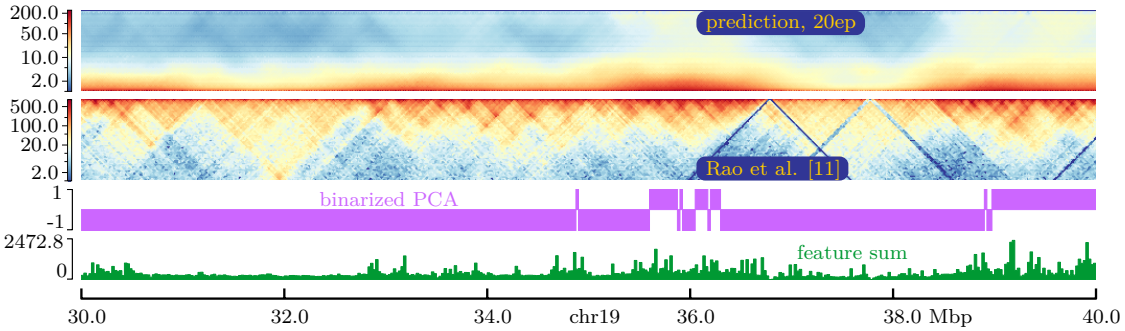
(f) Learning progress

Figure 52: Results / metrics cGAN, mixed embedding, no pre-training, $w = 64$, test chromosomes



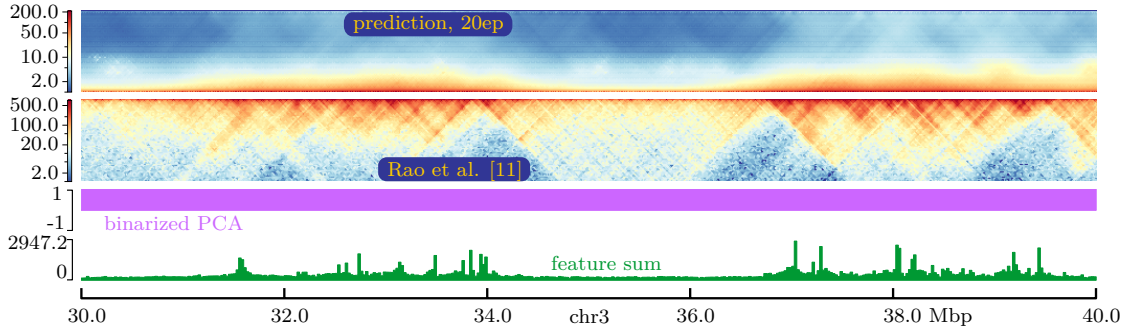
(a) Example region 1

see section 4.6 for track explanations



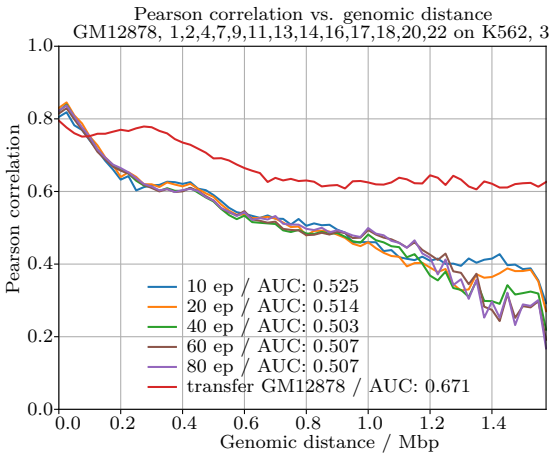
(b) Example region 2

see section 4.6 for track explanations

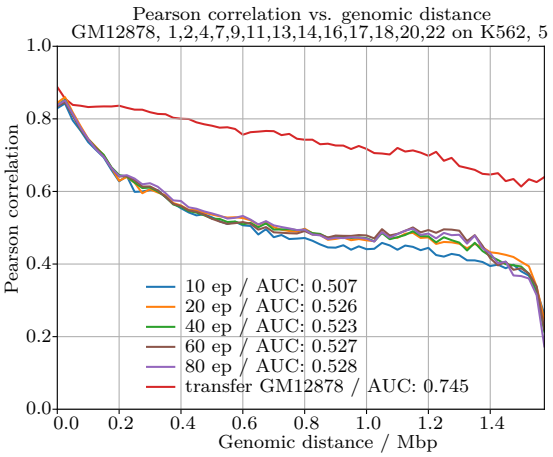


(c) Example region 3

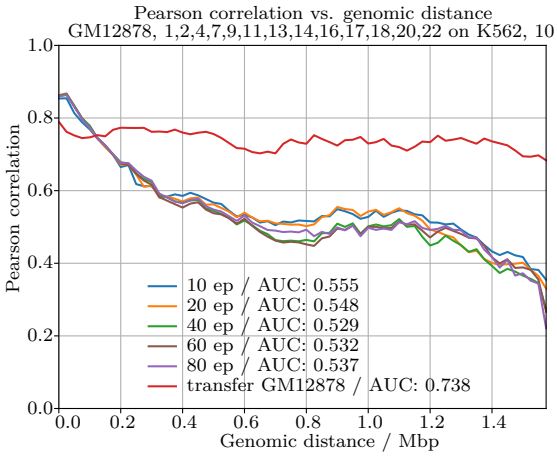
Figure 53: Example predictions GM12878 → K562, cGAN, mixed embedding, no pre-training, $w = 64$, 20 epochs



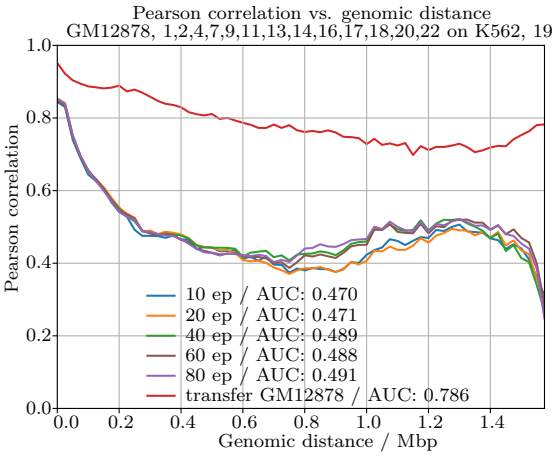
(a) chr3



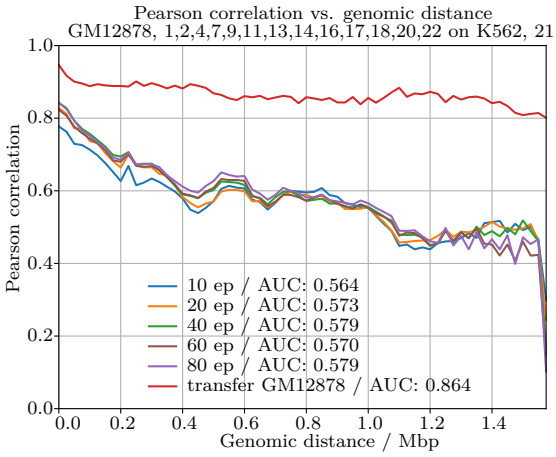
(b) chr5



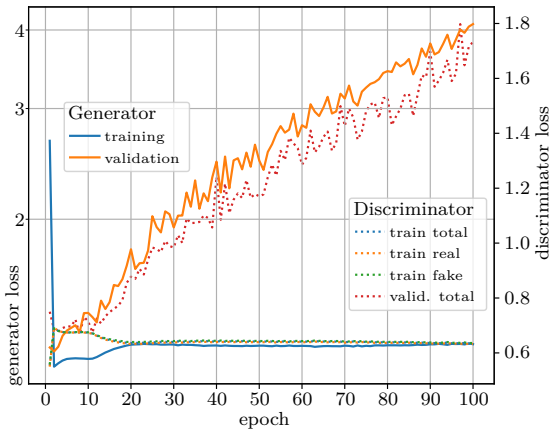
(c) chr10



(d) chr19

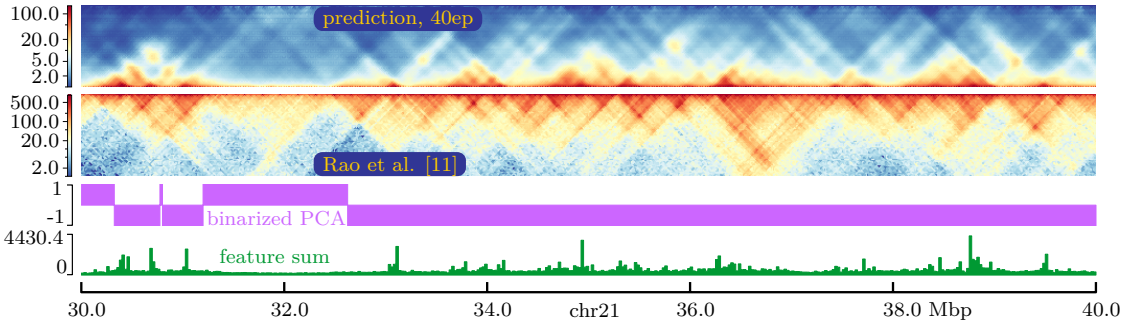


(e) chr21



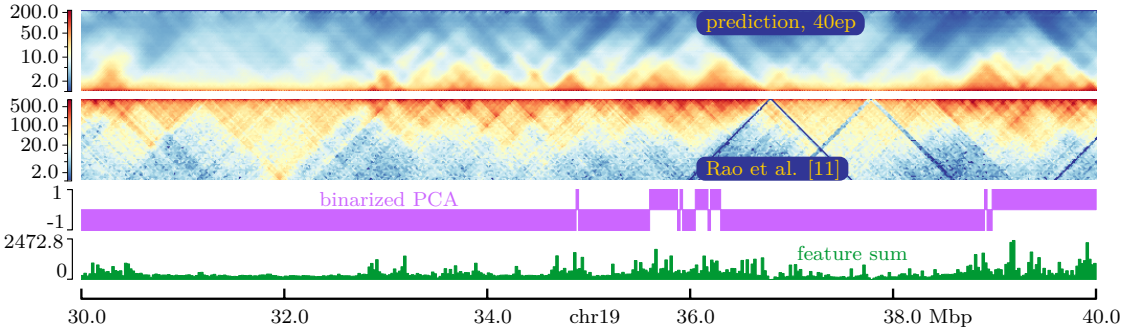
(f) Learning progress

Figure 54: Results / metrics cGAN, mixed embedding, DNN pre-trained, $w = 64$, test chromosomes



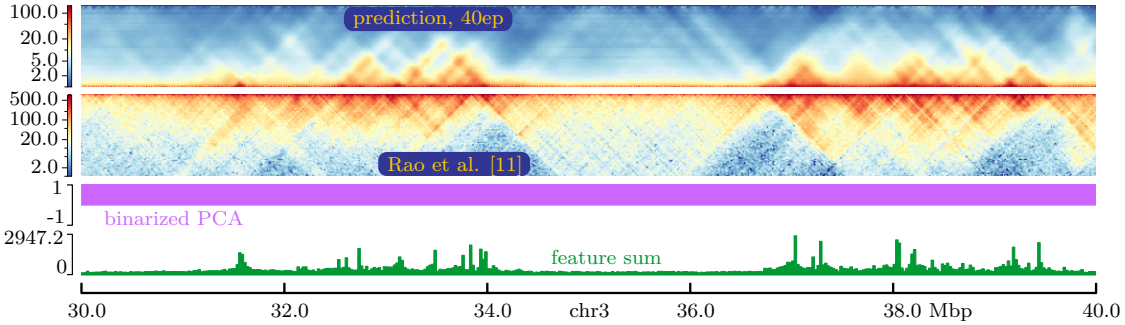
(a) Example region 1

see section 4.6 for track explanations



(b) Example region 2

see section 4.6 for track explanations



(c) Example region 3

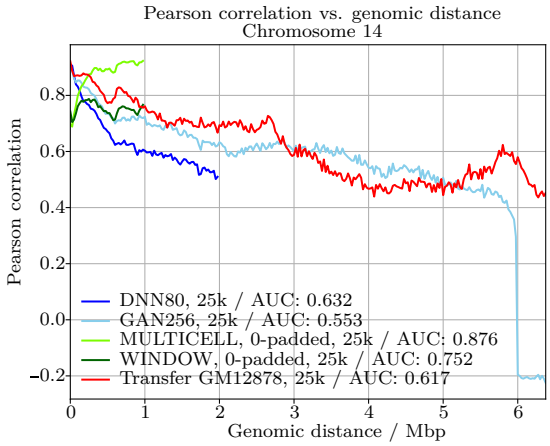
Figure 55: Example predictions GM12878 → K562, cGAN, mixed embedding, DNN pre-trained, $w = 64$, 40 epochs

5.3 Comparison with other approaches

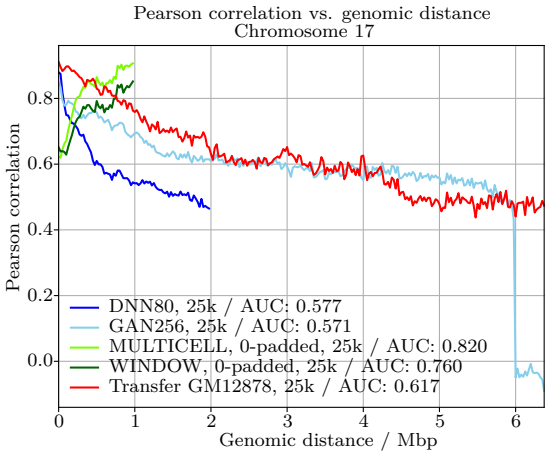
When comparing the random-forest-based method by Zhang et al., HiC-Reg [28], to the cGAN model trained on the typical training chromosome set, cf. 4.2, the cGAN approach seemed superior to all others for distances up to about 200 kbp, while both the multicell- and window-approach by Zhang et al. outperformed DNN and Hi-cGAN for distances between 200 kbp and 1 Mbp, fig. 56. This was also reflected in the matrix plots, fig. 57. While the cGAN often predicted smaller structures up to about 400 kbp very well and offered distinct boundaries even among nested structures, the approach by Zhang et al. showed better performance for interactions in the upper half of the windowsize, see e. g. chromosome 17, 30...34.5 Mbp. Note that the HiC-Reg WINDOW data stems from a random forest trained only on chromatin features and the Hi-C matrix from GM12878, while the HiC-Reg MULTICELL data has been obtained by training on chromatin feature data from GM12878, K562, HMEC, HUVEC and NHEK and a GM12878 matrix. Both HiC-Reg methods have used only data from chromosome 14 or 17 at binsize 5 kbp, while in this setting, Hi-cGAN has been trained on feature- and matrix data from GM12878, chromosomes 1, 2, 4, 7, 9, 11, 13, 14, 16, 17, 18, 20, and 22 at binsizes of 25 kbp.

To get a better comparison with the HiC-Reg WINDOW approach, in a second setting, Hi-cGAN was trained on data from chromosome 14 or 17 only. Surprisingly, despite the low amount of training samples, the training process converged with good Pearson correlations for the (training-)chromosomes and visually good matrices, fig. 56e/56f and 57c/57f. In general, Hi-cGAN was still better than HiC-Reg MULTICELL and WINDOW at smaller distances and worse at larger ones, but structures sized approximately 500 to 1000 kbp appeared more clearly and the "intersecting point" on the Pearson correlation graphs moved further to the right. Interestingly, interacting pairs at distances beyond approximately 1.3 kbp were all predicted zero, likely due to the small number of samples in this setting, maybe in combination with insufficient training. The effect was more pronounced for chromosome 17, which indeed yields less training samples.

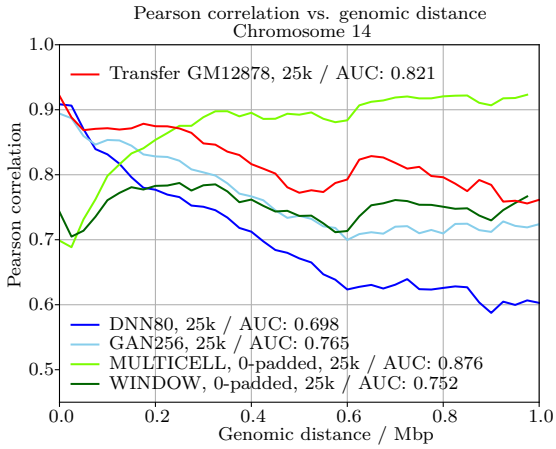
Additionally, we tried training our own implementation of HiC-Reg from a previous study project [13] on the same training data, binsizes and windowsizes as the cGAN model above to allow for a direct comparison. Unfortunately, we could not confirm the good results of HiC-reg, fig. 58 and 59, and it is currently unknown whether this was just due to our implementation or due to a general problem with the HiC-Reg approach in this setting. However, we have generally not been able to reproduce the results of HiC-Reg with our implementation so far.



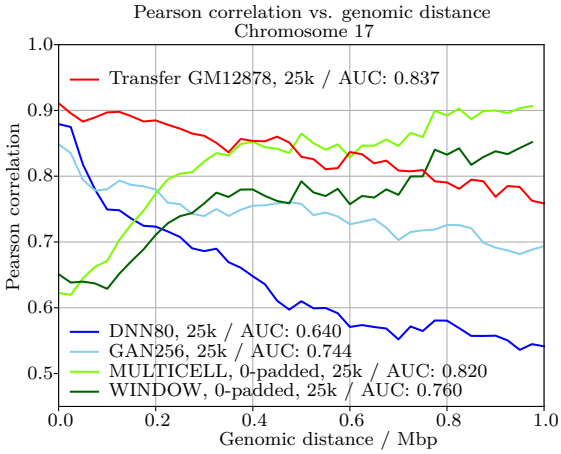
(a) Overview chromosome 14



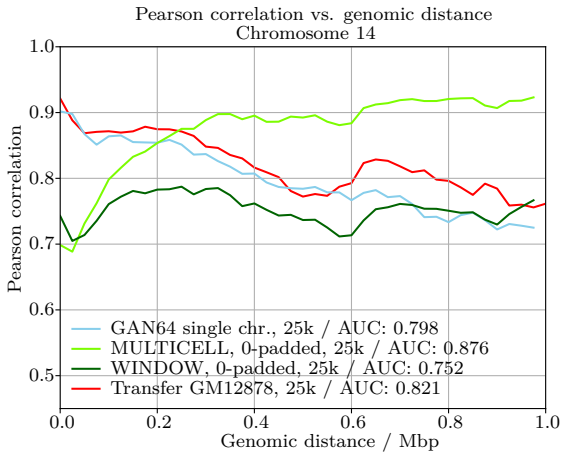
(b) Overview chromosome 17



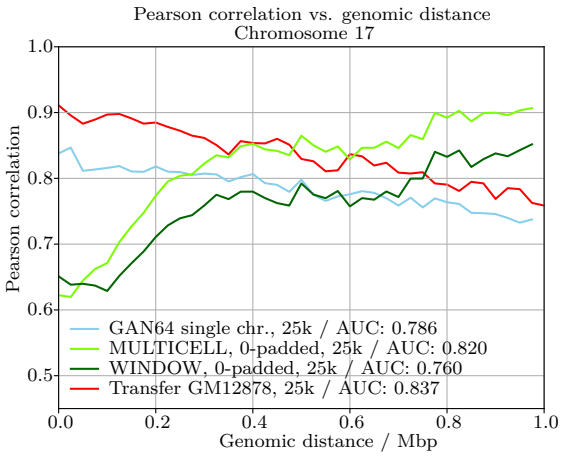
(c) chr14, detail 0...1 Mbp



(d) chr17, detail 0...1 Mbp

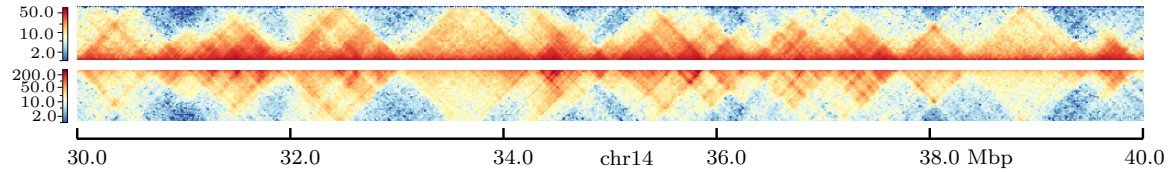


(e) chr14 cGAN single chrom

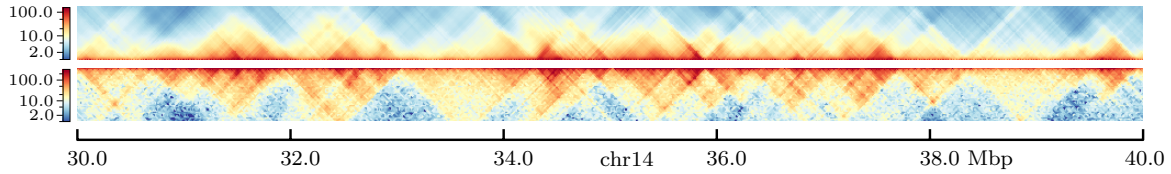


(f) chr17 cGAN single chrom

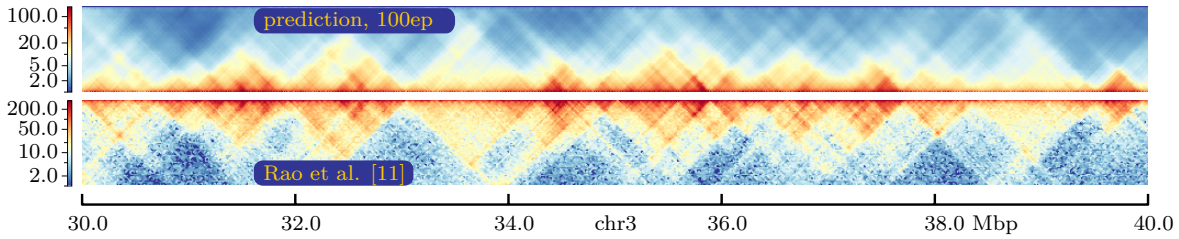
Figure 56: Pearson correlation comparison Hi-cGAN / DNN and HiC-Reg [28]



(a) Hic-Reg (random forest), MULTICELL, reconstructed from [28]

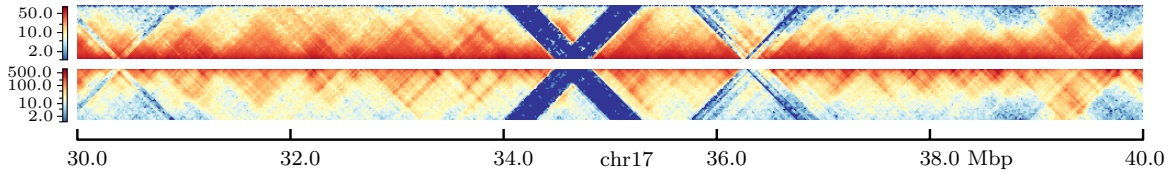


(b) Hi-cGAN with windowsize $w = 256$; plot restricted to first 1 Mbp

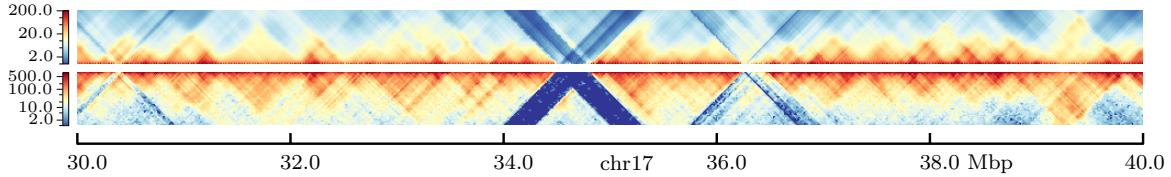


(c) Hi-cGAN with windowsize $w = 64$, trained on chr14 only

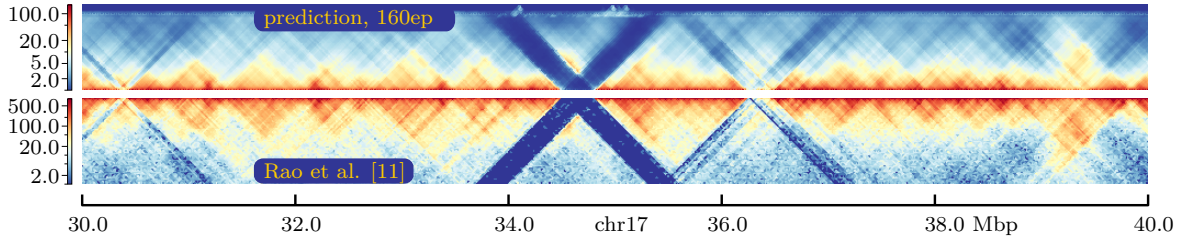
see section 4.6 for track explanations



(d) Hic-Reg (random forest), MULTICELL, chr17, reconstructed from [28]

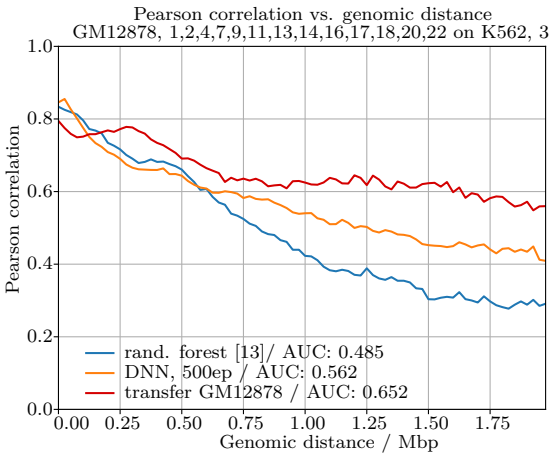


(e) Hi-cGAN with windowsize $w = 256$, chr17, plot restricted to first 1 Mbp

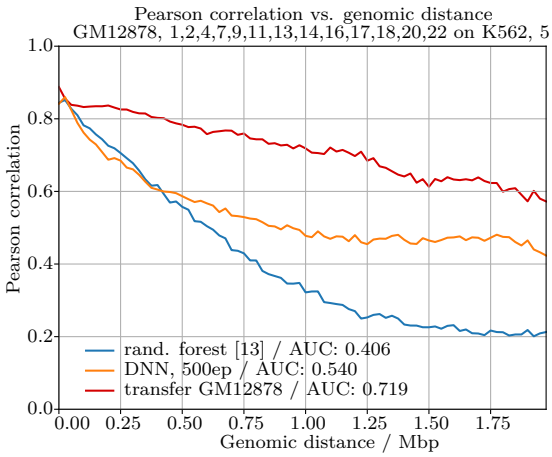


(f) Hi-cGAN with windowsize $w = 64$, trained on chr17 only

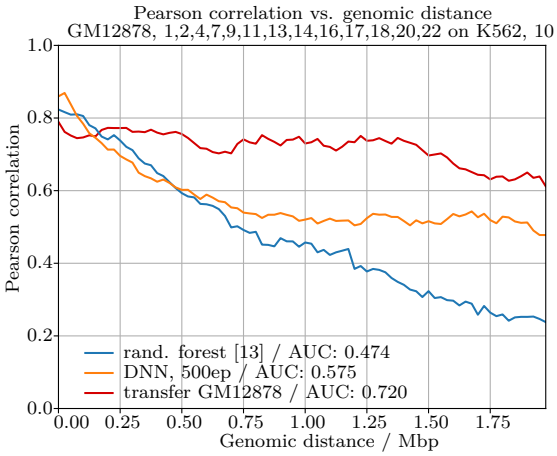
Figure 57: Comparison HiC-Reg [28] and Hi-cGAN (CNN embedding)



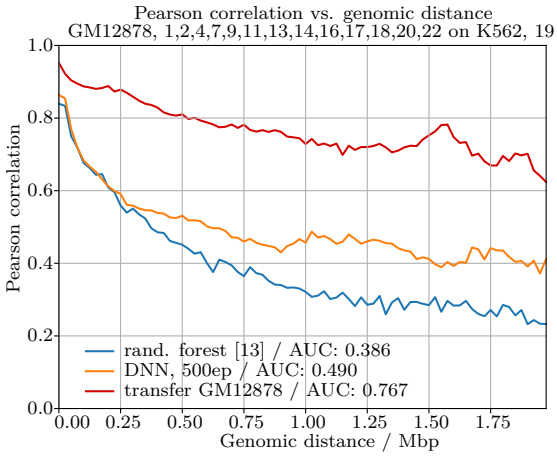
(a) chr3



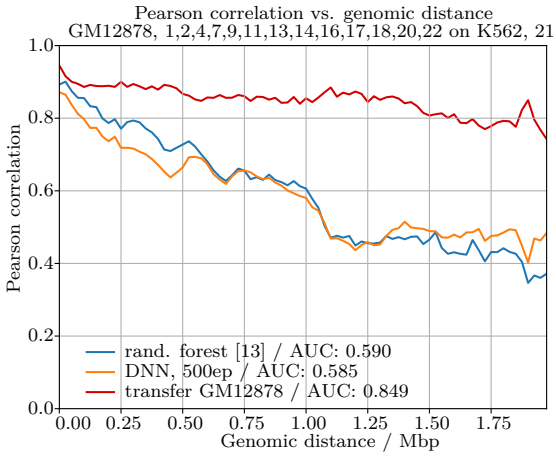
(b) chr5



(c) chr10

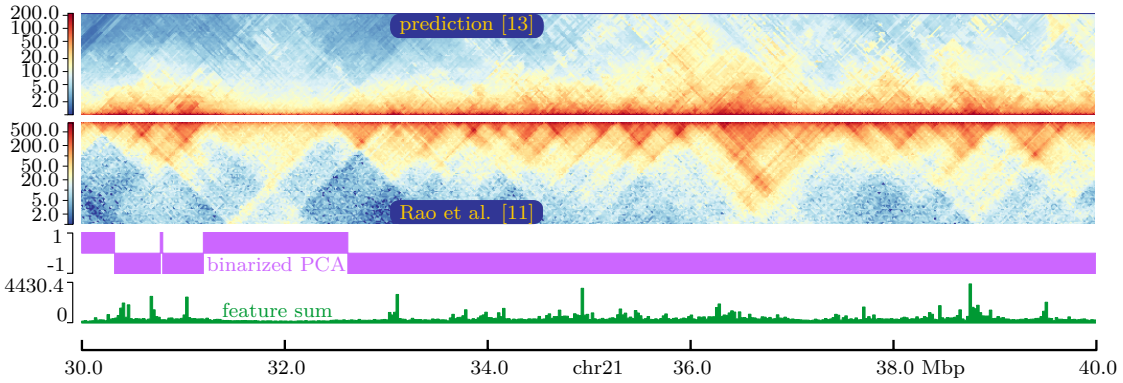


(d) chr19

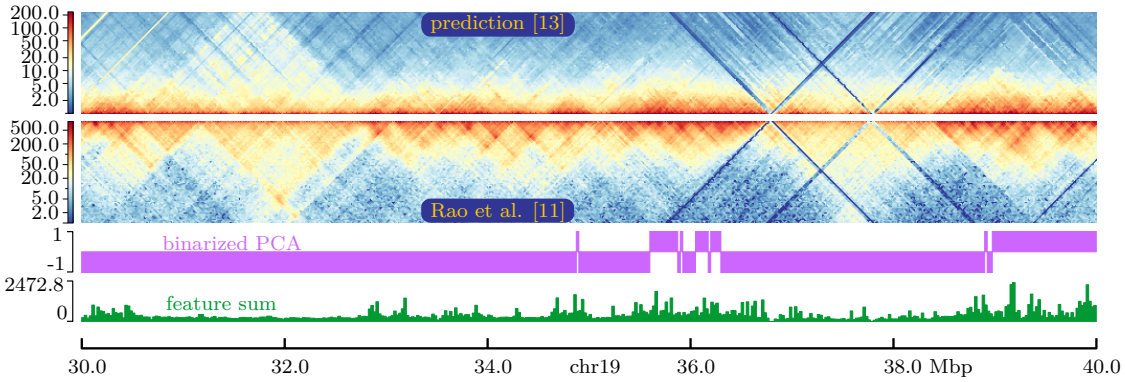


(e) chr21

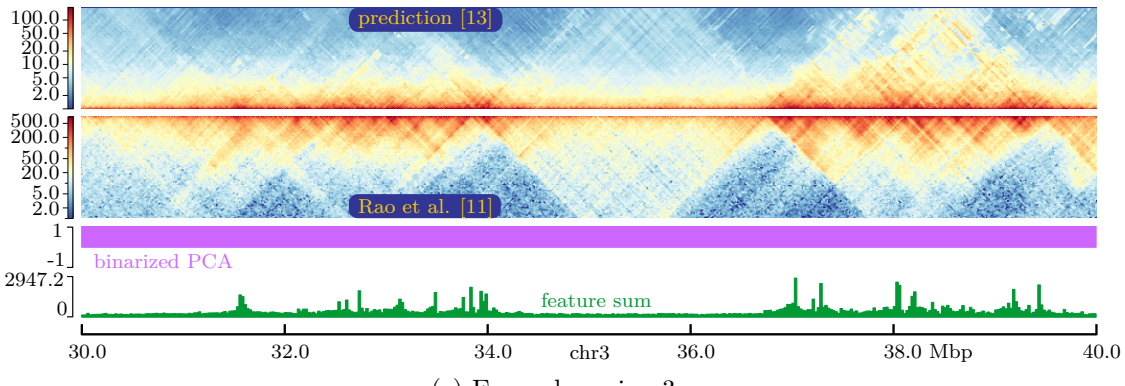
Figure 58: Results / metrics, random forest from study project [13], windowsize 80, 25 kbp, test chromosomes



(a) Example region 1



(b) Example region 2



(c) Example region 3

Figure 59: Example predictions GM12878 → K562, random forest [13], 25 kbp, $w = 80$

6 Discussion and Outlook

In this thesis, two methods have been considered for predicting Hi-C interaction matrices, one based on a dense neural network proposed by Farré et al. [30] and the other one based on a conditional generative adversarial network inspired by *pix2pix* [49].

Despite all amendments to the original network setup, the results of the dense neural network approach remained modest. Improvements, if at all visible, could only be achieved for a small part of the test set.

On the other hand, the novel conditional generative adversarial network (cGAN) method seems promising. Especially the variant with CNN embedding showed widely satisfying results for genomic distances up to about 5 Mbp and partially outperformed existing methods like *HiC-Reg* [28] with regard to predicting nested structures. However, there is still room for improvement – even the best predictions of this thesis were not or not much better than simply taking data from the training cell line as a "prediction" for the target cell line.

Throughout the thesis, a few starting points for further improving the predictions were identified. In the current implementation, the discriminator is trained on two kinds of inputs: it is supposed to classify pairs of true chromatin feature arrays and true Hi-C submatrices as real, and pairs of true chromatin feature arrays and artificial Hi-C submatrices produced by the generator as fake, cf. eq. (8). For text-to-image synthesis tasks, a third kind of input has been introduced; here the discriminator is additionally trained to classify pairs of true output and mismatching *input data* as fake [62]. Training the discriminator on such samples might help the cGAN to avoid situations where the generator produces matrices which "look real", but have insufficient correlations with the chromatin feature input data. It is unclear how strong the effect of this change would be in the given application, where the adversarial loss is only one part of the combined generator loss, but certainly worth a try.

As noted in section 4.8.1, better results were obtained by replacing the tanh activation in the generator of the original *pix2pix* network by sigmoid activation. Retrospectively, however, it might have been better to leave the tanh activation in the generator untouched and instead scale the training matrices – and maybe the chromatin features, too – to value range -1...1. This would allow for a broader input value range to the sigmoid function contained in the discriminator loss, potentially improving numerical gradient computations.

Another starting point is for sure the network architecture in itself. While good reasons backed the decision for choosing *pix2pix* for the general network setup, namely its wide range of applications and demonstrated convergence even when only few training samples are at hand [49], it remains a network designed for converting *input images* to output images. However, the task at hand essentially requires transforming *one-dimensional input* data into very special symmetric output images. Although surprisingly good results were obtained, it is difficult to imagine that *pix2pix* – amended by the simplistic and rather ad-hoc CNN embedding network described in section 3.2.3 and 4.8.3 – is the optimal solution for suchlike 1D $\rightarrow$ 2D conversions.

Nevertheless, the cGAN approach developed within this thesis can hopefully serve as a helpful starting point and baseline for further development of improved Hi-C contact matrix prediction methods. To this end, all source code is provided on github under an open source license [79, 80].

7 Appendix

7.1 Chromatin feature download details

The basic download paths for all chromatin feature files in .bam format are
`https://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeBroadHistone/`
`https://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeOpenChromDnase/`
`https://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeSydhTfbs/`
for CTCF/Histones, DNase, RAD21/SMC3, respectively. The actual files for replicates 1 and 2 are then easily be found by searching for "*CELL_FEATURE*", where *CELL* is the Cell line (e. g. GM12878) and *FEATURE* is the chromatin feature of interest, e. g. CTCF, H3K27ac and so on. For convenience, the pdf version of this document contains clickable links below.

GM12878 replicate 1	**GM12878 replicate 2**	**K562 replicate 1**	**K562 replicate 2**
CTCF	CTCF	CTCF	CTCF
H3k27ac	H3k27ac	H3k27ac	H3k27ac
H3k27me3	H3k27me3	H3k27me3	H3k27me3
H3k36me3	H3k36me3	H3k36me3	H3k36me3
H3k4me1	H3k4me1	H3k4me1	H3k4me1
H3k4me2	H3k4me2	H3k4me2	H3k4me2
H3k4me3	H3k4me3	H3k4me3	H3k4me3
H3k79me2	H3k79me2	H3k79me2	H3k79me2
H3k9ac	H3k9ac	H3k9ac	H3k9ac
H3k9me3	H3k9me3	H3k9me3	H3k9me3
H4k20me1	H4k20me1	H4k20me1	H4k20me1
DNase	DNase	DNase	DNase
Rad21	Rad21	Rad21	Rad21
SMC3	SMC3	SMC3	SMC3

feature	accession	feature	accession
BEAF	SRR1164521	H3K9me1	SRR869989
Chro Chriz WR	SRR869929	H3K9me2	SRR869708
CP190-HB	SRR869732	H3K9me2	SRR869864
CTCF	SRR869933	H3K9me3	SRR869860
dMi	SRR869937	H4	SRR870049
dRING	SRR869941	H4K16acM	SRR869856
GAF	SRR869740	H4K20me1	SRR870057
H1	SRR870123	HP1	SRR869718
H2AV	SRR869950	HP1b	SRR870061
H2B-ubiq	SRR869956	HP1c	SRR870183
H3	SRR869965	HP2	SRR870138
H3K18Ac	SRR869977	HP4	SRR869880
H3K23ac	SRR869981	JHDM1	SRR870069
H3K27Ac	SRR869744	LSD1	SRR870073
H3K27me2	SRR869993	MBD-R2	SRR870077
H3K27me3	SRR869712	MOF	SRR870081
H3K36me1	SRR869839	NURF301	SRR870085
H3K36me2	SRR870231	Pc	SRR869724
H3K36me3	SRR869900	POF	SRR870089
H3K4me1	SRR870009	Psc	SRR869736
H3K4me3	SRR870025	RNA Pol II	SRR870093
H3K79Me1	SRR869748	RPD3	SRR870105
H3K79me2	SRR869884	SU(HW)-HB	SRR869728
H3K79me3	SRR870191	ZW5	SRR870117
H3K9acS10P	SRR870033		

Table 5: Chromatin features from D. melanogaster used for the comparison DNN / cGAN / Farre et al. [30]

7.2 Listings

```
#bash-style code
#convert from hic to cooler, single resolution
#MATRIXHIC is a matrix in .hic format
hic2cool convert -r 5000 $MATRIXHIC matrix_5k.cool
#coarsen the matrix from 5k to 25k, for example
cooler coarsen -k 5 matrix_5k.cool -o matrix_25k.cool
#versions used for thesis
#hic2cool 0.8.3, cooler 0.8.10
```

Listing 1: Hic to cooler

```
#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}
#versions used for thesis
#samtools 1.9, bamCoverage 3.5.0, bigwigCompare 3.5.0
```

Listing 2: Bam to bigwig

```
chromosome="${2}"
bamfolder="${1}"

shopt -s nullglob
for i in ${1}*.bam
do
 echo "converting: ${i} - chrom ${2}"
 tmpfile="${i}.tmp"
 countfile="${i%.bam}_${chromosome}.count"
 samtools view -b ${i} ${chromosome} > ${tmpfile}
 cmd="-ibam ${tmpfile} -bg"
 bedtools genomecov ${cmd} > ${countfile}
 rm ${tmpfile}
done
```

Listing 3: Bam to bedgraph

```
inputfolder="${1}"
#following example files are provided in HiC-Reg github
region="HiC-Reg/Scripts/aggregateSignalInRegion/hg19_5kbp_chr17.txt"
sizesfile="HiC-Reg/Scripts/aggregateSignalInRegion/hg19.fa.fai"

shopt -s nullglob
for i in ${1}*.count
do
 echo "converting: ${i}"
 outfile="${i%.count}.txt"
 cmd="${region} ${sizesfile} ${i} ${outfile}"
 ./aggregateSignal_v1 ${cmd} #provided by HiC-Reg
done
```

Listing 4: Bedgraph to HiC-Reg input format

```
#"combined_30" GM12878 or K562 matrix from Rao et al. at 5kb resolution
inMatrix=${1}
#filename for cooler matrix (*.cool) with VC_SQRT correction
outMatrix=${2}
chrom=${3}

cmd="-m ${inMatrix}"
cmd="${cmd} -o ${outMatrix}"
cmd="${cmd} --inputFormat cool --outputFormat cool"
cmd="${cmd} --correction_name VC_SQRT --store_applied_correction"
cmd="${cmd} --correction_division"
cmd="${cmd} --chromosome ${chrom}"
hicConvertFormat ${cmd}

cooler dump ${outMatrix} --join -o ${outMatrix%.cool}.txt
params="--infile ${outMatrix%.cool}.txt"
params="${params} --outfile ${outMatrix%.cool}_converted.txt"
python matrix_conversion.py ${params} #see below for details
```

Listing 5: Cooler matrices to HiC-Reg input format

```
import click
import pandas as pd

@click.option("--infile", required=True, type=click.Path(exists=True,
    readable=True, dir_okay=False), help="text file created by cooler dump
     with --join")
@click.option("--outfile", required=False, type=click.Path(writable=True,
     dir_okay=False))
@click.command()
def convert(infile, outfile):
  if outfile is None:
    outfile = infile
  try:
    df = pd.read_csv(infile, header=None, sep="\t", index_col=False)
  except Exception as e:
    msg = str(e)
    msg += "\ncould not read csv file, wrong format?"
    raise SystemExit(msg)
  if df.shape[1] != 7:
    msg = "Read file with wrong format, number of columns should be 7 but
     is {:d}"
```

```
18      msg = msg.format(df.shape[1])
19      raise SystemExit(msg)
20    #check if the matrix contains nans
21    if df.isnull().values.any():
22      msg = "WARNING: there are nans in the dataframe"
23      print(msg)
24    #create the text-encoded matrix format for HiC-Reg
25    df["start"] = "chr" + df.loc[:,0].astype(str) + "_" + df.loc[:,1].
        astype(str) + "_" + df.loc[:,2].astype(str)
26    df["end"] = "chr" + df.loc[:,3].astype(str) + "_" + df.loc[:,4].astype(
        str) + "_" + df.loc[:,5].astype(str)
27    df["count"] = df.loc[:,6].astype("float32")
28    df.drop(columns=[0,1,2,3,4,5,6], inplace=True)
29    df["count"].fillna(value=0.0, inplace=True)
30    #remove matrix diagonal since it is missing in the HiC-Reg examples,
        too
31    diagFilter = df["start"] == df["end"]
32    df = df.loc[~diagFilter]
33
34    df.to_csv(outfile, sep="\t", columns=["start","end","count"], header=
        False, index=False, float_format="%.4f")
35    print("first few lines of matrix txt file:\n", df.head())
36
37  if __name__ == "__main__":
38    convert()
```

Listing 6: Custom text file conversion for HiC-Reg's matrix input

```
1  import click
2  import pandas as pd
3  import cooler
4  import os
5  import numpy as np
6
7  @click.option("--filenames", "-f", required=True,
8          type=click.Path(exists=True, readable=True, dir_okay=False),
9          multiple=True,
10         help="testset_error file(s) from HiC-Reg")
11 @click.option("--chromsize", "-cs", required=False,
12         type=click.IntRange(min=1),
13         help="Size of chromosome. Will be derived from data, if not
       specified")
14 @click.option("--outfolder", "-o", required=True,
15         type=click.Path(writable=True, file_okay=False, exists=True),
16         help="Folder where the coolers will be placed")
17 @click.option("--exponentiate", "-e", required=False, type=bool, default=
       False, show_default=True, help="exponentiate count values")
18 @click.command()
19 def reconstruct(filenames, chromsize, outfolder, exponentiate):
20   #read the the text files from HiC-Reg
21   #skip the first row, since it contains the incomplete header with
       missing "Distance"
22   try:
23     dataframes = [pd.read_csv(filename, names=["Pair", "TrueValue", "
       PredictedValue", "SquaredErr", "Distance"],
24                   sep="\t",
25                   dtype={"Pair": str, "TrueValue": float, "PredictedValue
       ": float, "SquaredErr": float, "Distance": int},
```

```
                skiprows=[0]) for filename in filenames]
except Exception as e:
  msg = str(e) + "\nCould not read one of the files, wrong format?"
  raise SystemExit(msg)

#print number of pairs in each file
for i, df in enumerate(dataframes):
  print("file {:d} - {:d} pairs ({:s})".format(i, df.shape[0],
  filenames[i]))

for df in dataframes:
  #extract the start- and end positions of all pairs and the chromosome
  df["Pair"] = df["Pair"].str.replace(pat="-", repl="_")
  splitDf = df["Pair"].str.split(pat="_", expand=True)
  df["chrom"] = splitDf.iloc[:,0]
  df["start_L"] = splitDf.iloc[:,1].astype("int32")
  df["end_L"] = splitDf.iloc[:,2].astype("int32")
  #df["chr_R"] = splitDf.iloc[:,3] #not needed here, always the same as
   chr_L
  df["start_R"] = splitDf.iloc[:,4].astype("int32")
  df["end_R"] = splitDf.iloc[:,5].astype("int32")
  #compute binsizes
  df["binsizes"] = df["end_L"] - df["start_L"]
  #compute the bin ids
  binsize = df["binsizes"].iloc[0]
  df["bin1_id"] = df["start_L"] // binsize
  df["bin2_id"] = df["start_R"] // binsize
  #if desired, take the values to the power of e to get counts from log
  -scale values
  if exponentiate == True:
    df["TrueValue"] = np.exp(df["TrueValue"])
    df["PredictedValue"] = np.exp(df["PredictedValue"])
  #drop unnecessary columns
  df.drop(columns=["Pair", "Distance", "SquaredErr", "end_L", "end_R",
  "start_L", "start_R"], inplace=True)

#check if the binsizes are equal in the single dataframes before
  merging the dataframes
binsizes = [df["binsizes"].iloc[0] for df in dataframes]
binsizes = list(set(binsizes))
if len(binsizes) != 1:
  msg = "Binsizes not equal: {:s}".format(", ".join([str(b) for b in
  binsizes]))
  raise SystemExit(msg)
else:
  print("detected binsize: {:d}".format(binsizes[0]))

#concat the dataframes from the files
joinDf = pd.concat(dataframes, ignore_index=True)
joinDf.drop(columns=["binsizes"], inplace=True)

#When using train_i and test_i, there should be no duplicate pairs.
#Otherwise, e.g. in x-validation settings, take the mean
chrom = joinDf["chrom"].iloc[0]
elem_number_before = joinDf.shape[0]
joinDf = joinDf.groupby(["bin1_id", "bin2_id"])[["TrueValue", "
  PredictedValue"]].mean().reset_index()
```

```
76  elem_number_after = joinDf.shape[0]
77  if elem_number_after != elem_number_before:
78    msg = "Aggregated {:d} duplicates by taking the mean".format(
      elem_number_before - elem_number_after)
79    print(msg)
80
81  print("first few elems of joint dataset:\n", joinDf.head())
82  print("dataset contains {:d} pairs".format(joinDf.shape[0]))
83
84  #if no data given, set chromsize to max(bin id) * bin size + bin size
85  maxbin = joinDf[["bin1_id","bin2_id"]].max().max()
86  if chromsize is None:
87    chromsize = maxbin * binsizes[0] + binsizes[0]
88
89  #prepare the bins for cooler
90  bins = pd.DataFrame(columns=['chrom','start','end'])
91  binStartList = list(range(0, chromsize, binsizes[0]))
92  binEndList = list(range(binsizes[0], chromsize, binsizes[0])) + [
      chromsize]
93  bins['start'] = np.uint32(binStartList)
94  bins['end'] = np.uint32(binEndList)
95  bins["chrom"] = str(chrom)
96
97  #prepare the pixels for cooler
98  joinDf.sort_values(by=["bin1_id", "bin2_id"], inplace=True)
99  pixels = joinDf[["bin1_id","bin2_id"]].copy()
100 pixels["count"] = joinDf["PredictedValue"]
101 outfilename = os.path.join(outfolder, "predValues_{:s}.cool".format(str
      (chrom)))
102 #build the coolers
103 cooler.create_cooler(outfilename,
104           bins=bins,
105           pixels=pixels,
106           dtypes={'count': np.float64},
107           ordered=True,
108           metadata={"fromFilenames": filenames})
109 pixels["count"] = joinDf["TrueValue"]
110 outfilename = os.path.join(outfolder, "trueValues_{:s}.cool".format(str
      (chrom)))
111 cooler.create_cooler(outfilename,
112           bins=bins,
113           pixels=pixels,
114           dtypes={'count': np.float64},
115           ordered=True,
116           metadata={"fromFilenames": filenames})
117 #check and report sparsity
118 max_bin_dist = (pixels["bin2_id"] - pixels["bin1_id"]).max()
119 full_nr_elements = ((bins.shape[0])**2 + bins.shape[0])//2
120 cut_nr = bins.shape[0] - max_bin_dist
121 cut_nr_elements = (cut_nr**2 + cut_nr)//2
122 exp_nr_elements = full_nr_elements - cut_nr_elements
123 sparsity_percent = (pixels.shape[0] / exp_nr_elements)*100
124 print("number of bins: {:d}".format(bins.shape[0]))
125 print("maxdist: {:d}".format(max_bin_dist * binsizes[0]))
126 print("expected interacting pairs: {:d}, available interacting pairs:
      {:d}".format(exp_nr_elements, pixels.shape[0]))
127 print("sparsity: {:.2f}%".format(sparsity_percent))
```

```
128
129 if __name__ == "__main__":
130   reconstruct()
```

Listing 7: Custom script to convert HiC-Reg's text output to cooler format

```
1  fqdir=${1}                  #directory with fastq files
2  procs="8"                   #number of threads
3  indexdir="./BDGP5/BDGP5"    #bowtie2 index for BDGP5/dm3
4
5  shopt -s nullglob
6  for fqfile in ${fqdir}*.fastq; do
7    samfile="${fqfile%fastq}sam"
8    bamfile="${fqfile%fastq}bam"
9    echo "processing ${fqfile} => ${samfile} => ${bamfile}"
10   bowtie2 -x ${indexdir} -U ${fqfile} -S ${samfile} --no-unal -p ${procs}
11   samtools view -S -b ${samfile} > ${bamfile}
12   rm ${samfile}
13   gzip ${fqfile}
14   samtools sort ${bamfile} -o ${bamfile}
15   samtools index ${bamfile}
16 done
```

Listing 8: Mapping ChIP-seq reads to D. melanogaster ref. genome

7.3 Hardware

For the thesis, three virtual machines were used to train the neural networks, see table 6. All training for section 5.2 with windowsize $w = 256$ was done on machine 2, while computations for section 5.2 with windowsizes $w \in \{64, 128\}$ were done on machine 1. Computations for sections 5.1.1, 5.1.2 and 5.1.5 were done on machine 3 without GPU and computations for sections 5.1.3 and 5.1.4 were done on machine 2.

Training the DNN with perception- or score-based losses as well as the cGAN was not reasonably possible without GPU. For the cGAN, it was found that GPU memory should not fall short of the given values (table 6) to avoid undue limitations on batchsizes and/or windowsizes. 20 GB of main memory, as in machine 2, were not enough to train the cGAN at windowsize 64 for more than 100 epochs. It could not be clarified throughout the thesis whether this was due to a memory leak in tensorflow or due to the chosen implementation.

Training samples were stored as tensorflow tfrecords (on the fly at runtime) and a custom pipeline including a shuffle buffer and prefetching was employed to balance workload between CPUs and GPU. This approach was found to be faster than generator-based approaches by a large margin. Note: CPU frequency in GHz; L1,L2,L3 cache in KB; GPU memory in MiB; RAM (main memory) in GB

	CPU					**GPU**		**RAM**
machine	#	make/model	freq.	L1	L2	L3	make/model	memory	in GB
1	8	AMD EPYC 7742 64-Core	2.25	64	512	16384	NVIDIA Tesla T4 TU104GL	15109	20
2	40	Intel Xeon E5-2630 v4	2.20	32	4096	16384	NVIDIA Tesla T4 TU104GL	15109	116
3	20	AMD EPYC 7351P 16-Core	2.40	1300	10000	320000	–	–	116

Table 6: Key figures of hardware used throughout the thesis

7.4 Further figures

7.4.1 Combined loss function

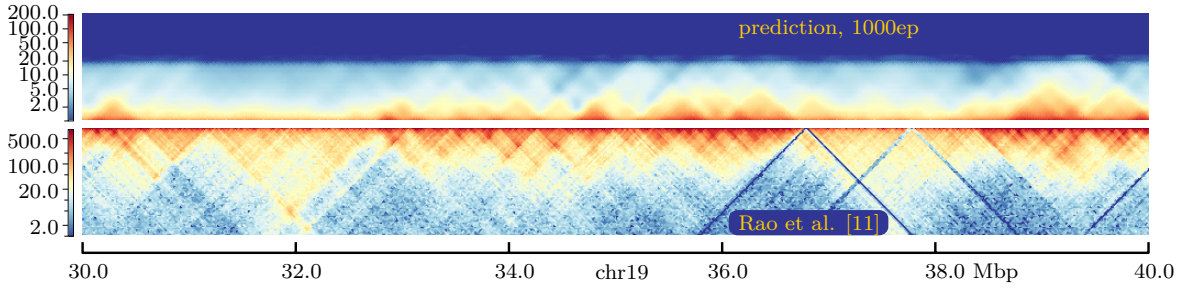
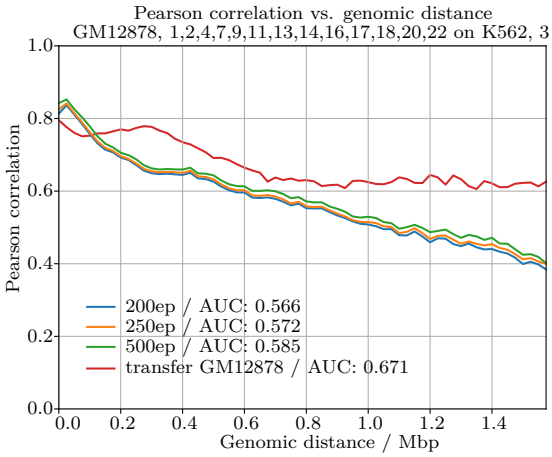
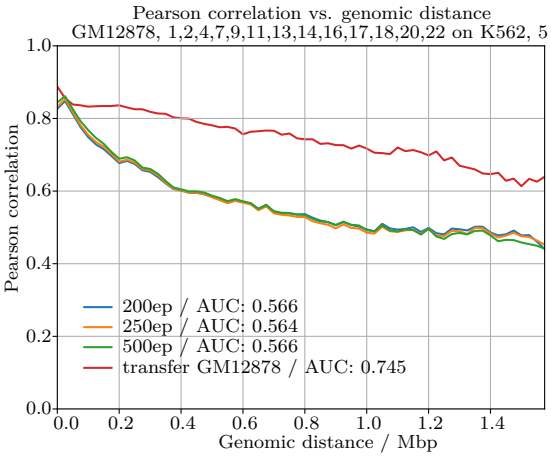


Figure 60: Combined loss with parametrization $\lambda_{MSE} = 10^{-5}$, $\lambda_{VGG} = 0.0$, $\lambda_{TV} = 1.0$

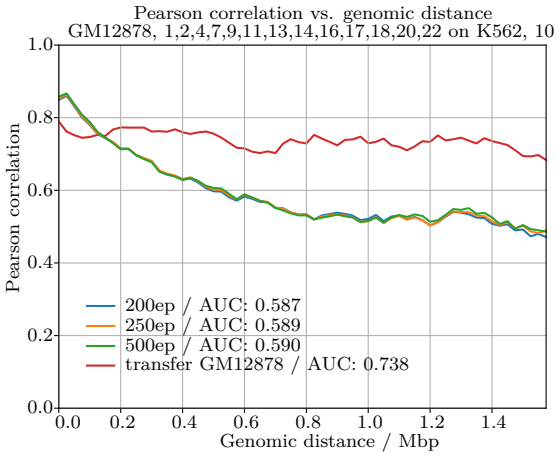
7.4.2 Results of pre-training the DNN-embedding



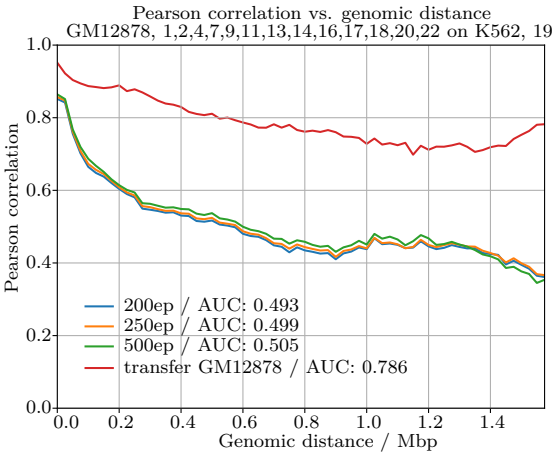
(a) chr3



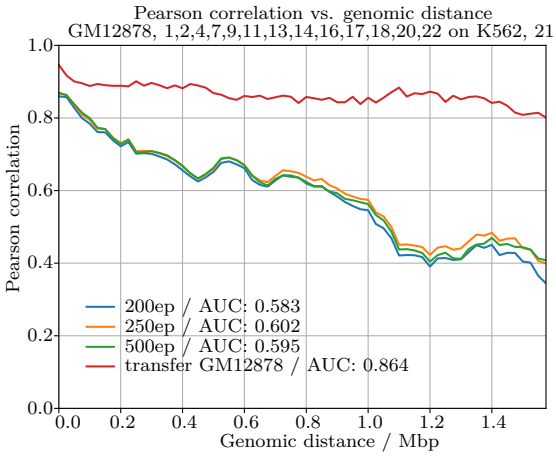
(b) chr5



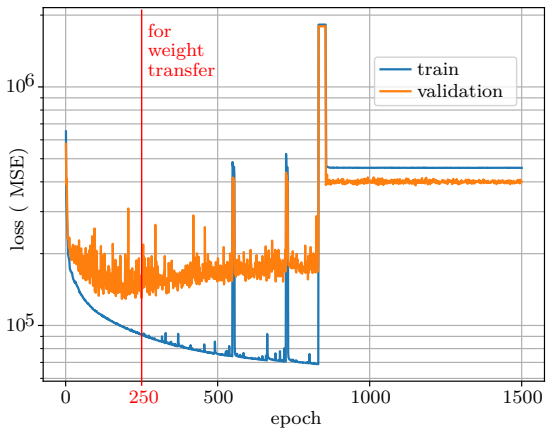
(c) chr10



(d) chr19

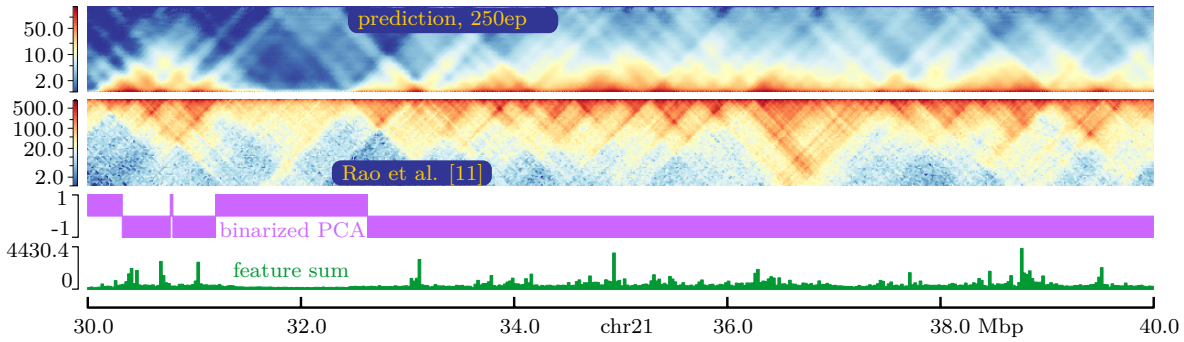


(e) chr21



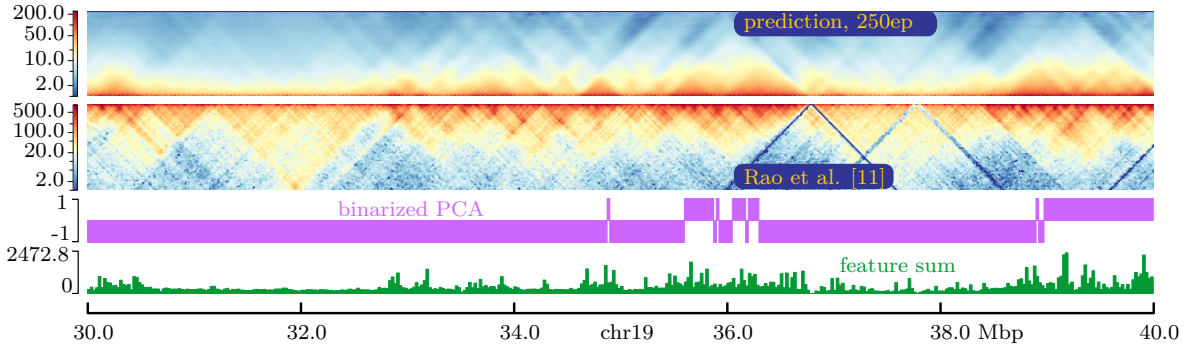
(f) Learning progress

Figure 61: Results / metrics DNN, $w = 64$, test chromosomes



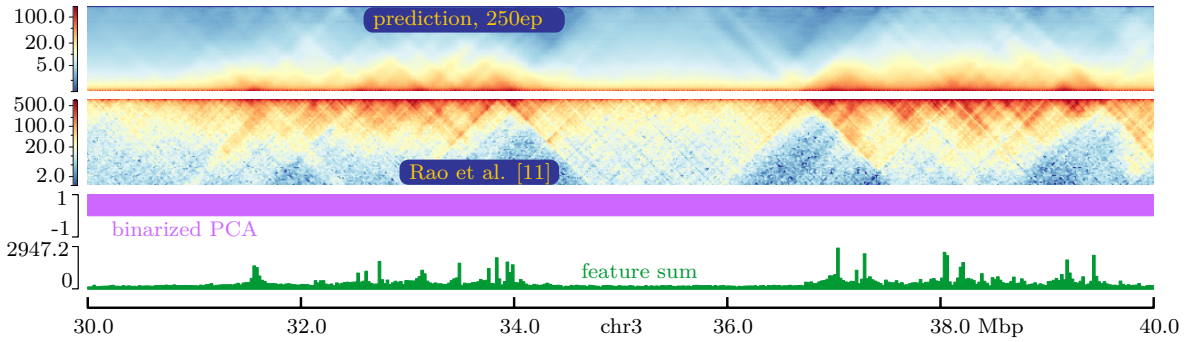
(a) Example region 1

see section 4.6 for track explanations



(b) Example region 2

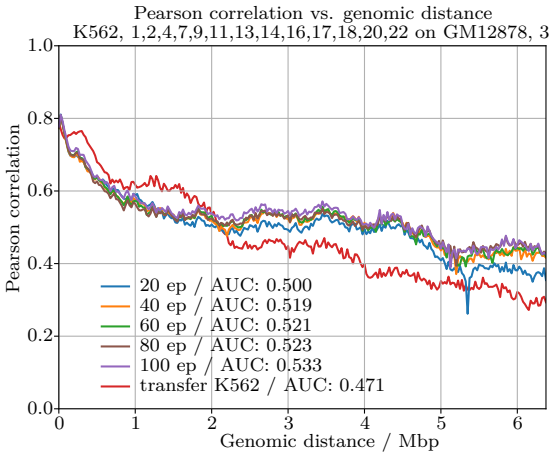
see section 4.6 for track explanations



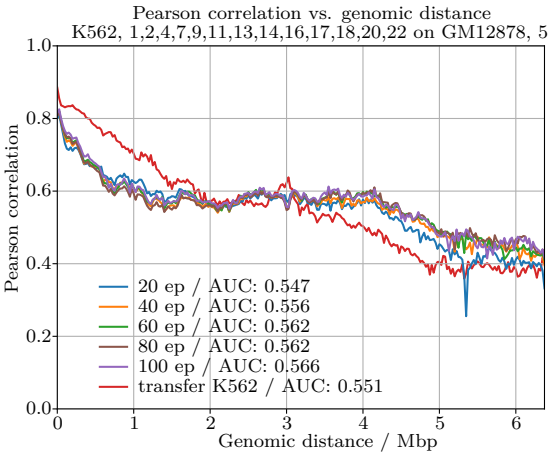
(c) Example region 3

Figure 62: Example predictions GM12878 → K562, DNN, $w = 64$, 250 epochs

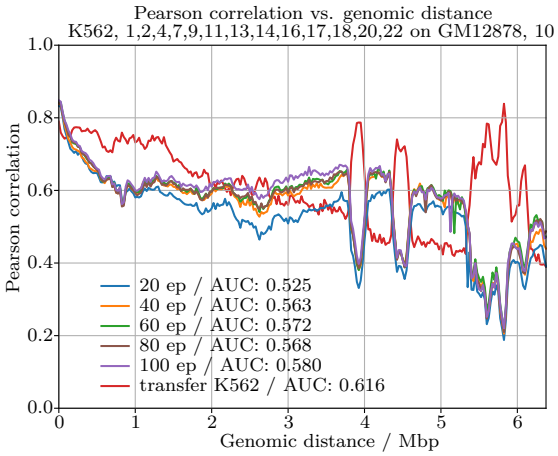
7.4.3 cGAN trained on K562, predicting GM12878



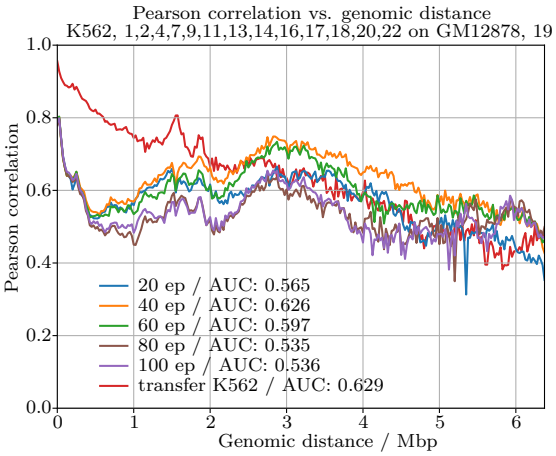
(a) chr3



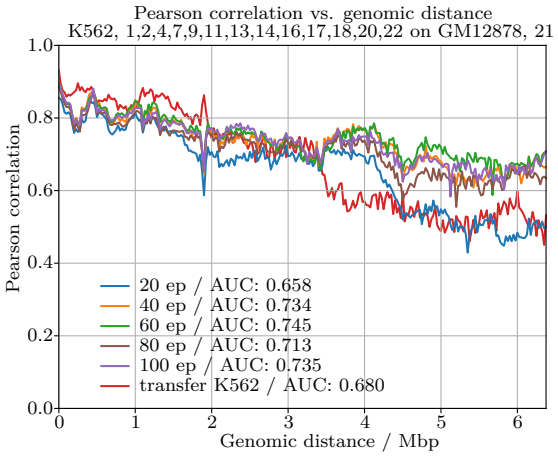
(b) chr5



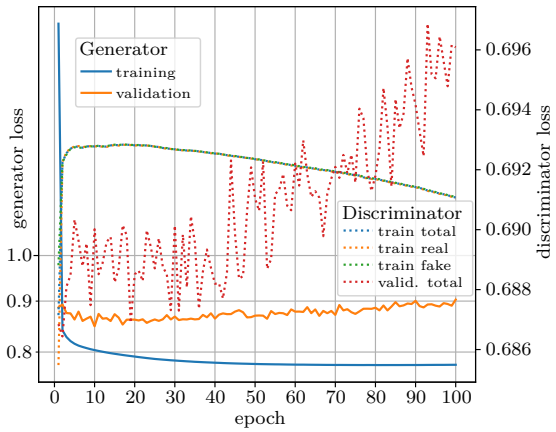
(c) chr10



(d) chr19

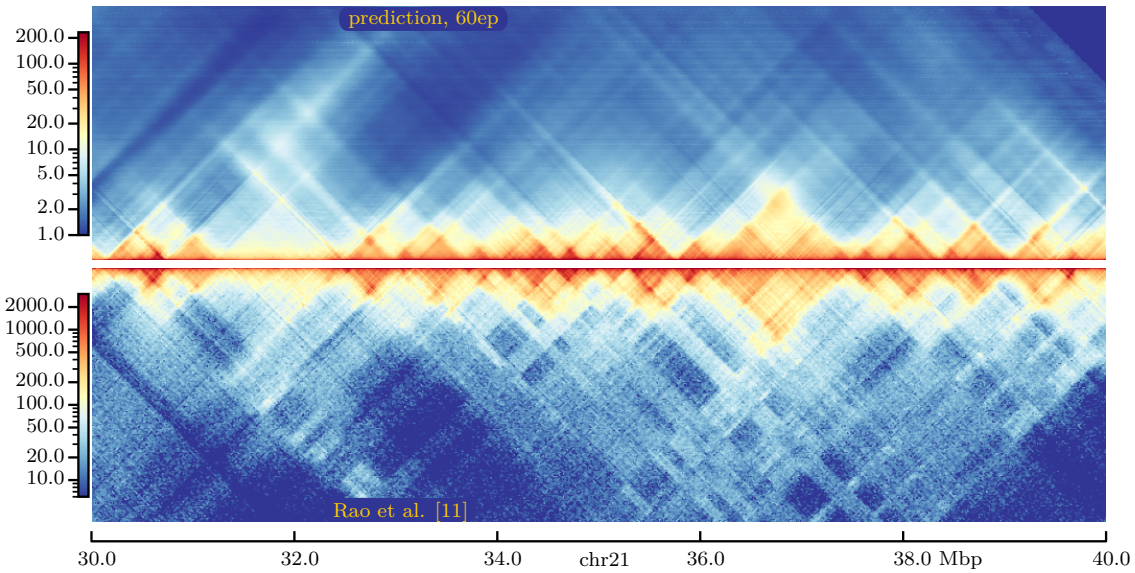


(e) chr21

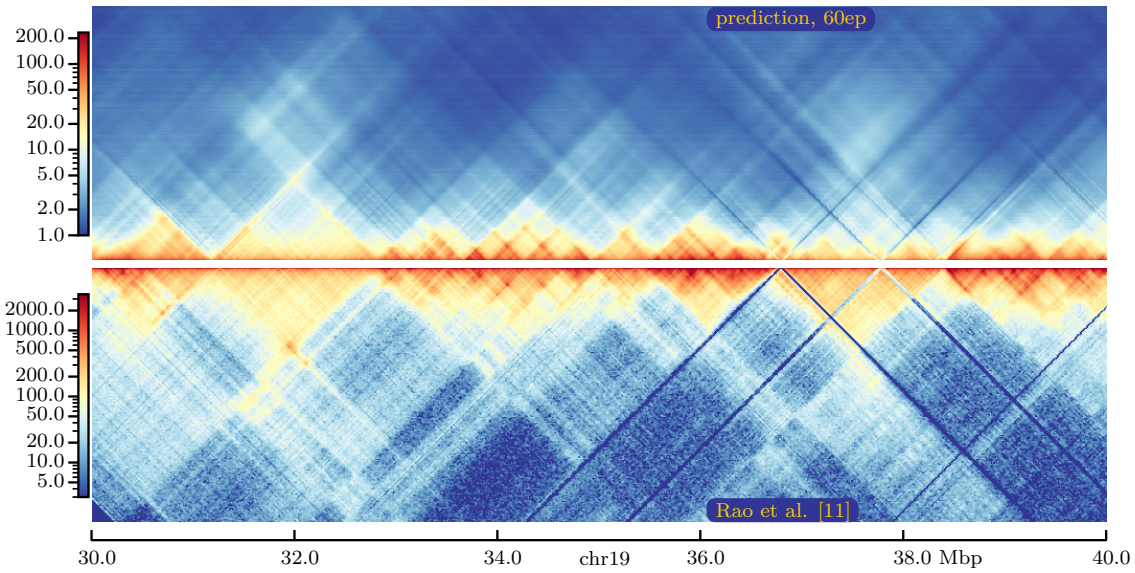


(f) Learning progress

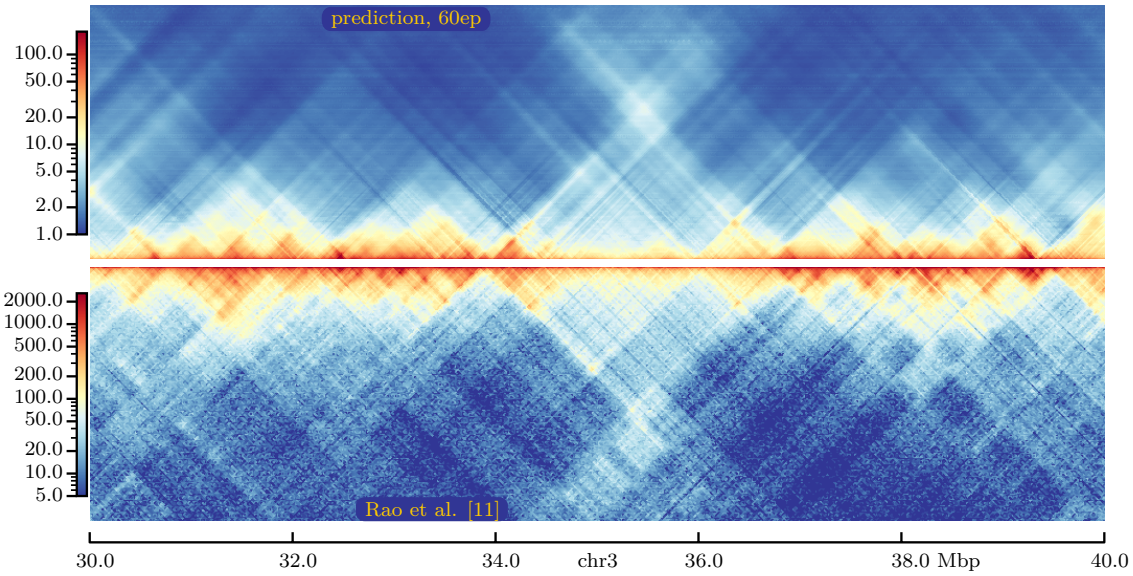
Figure 63: Results / metrics cGAN, CNN embedding, $w = 256$, test chromosomes



(a) Example region 1



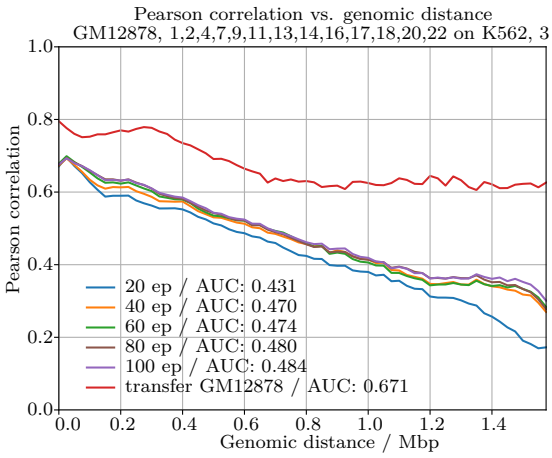
(b) Example region 2



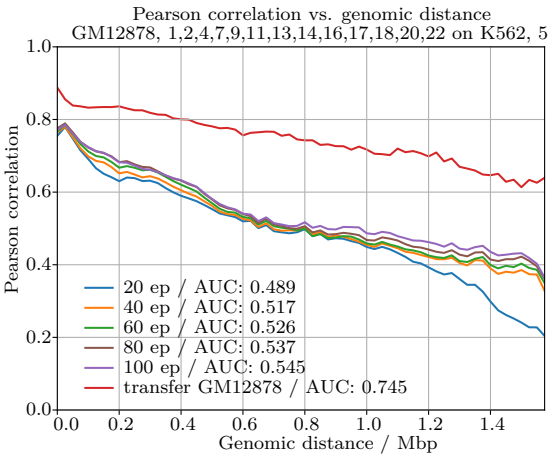
(c) Example region 3

Figure 64: Example predictions K562 → GM12878, cGAN, CNN embedding, $w = 256$, 60 epochs

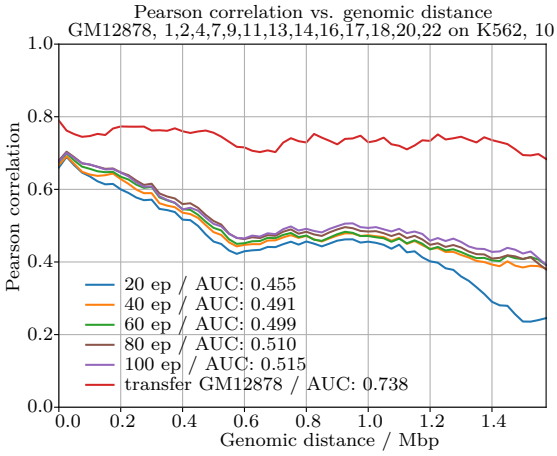
7.4.4 cGAN trained on single chromosomes predicting usual test chromosomes



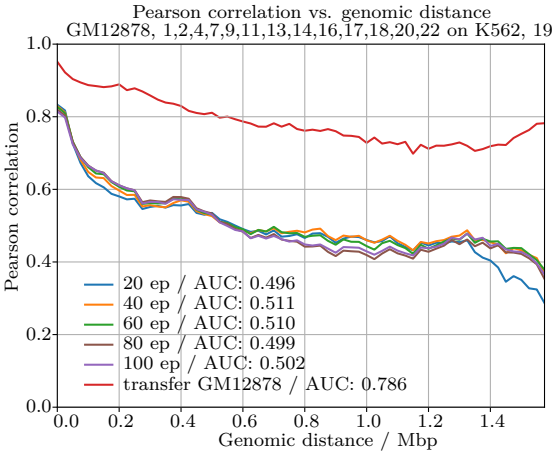
(a) chr3



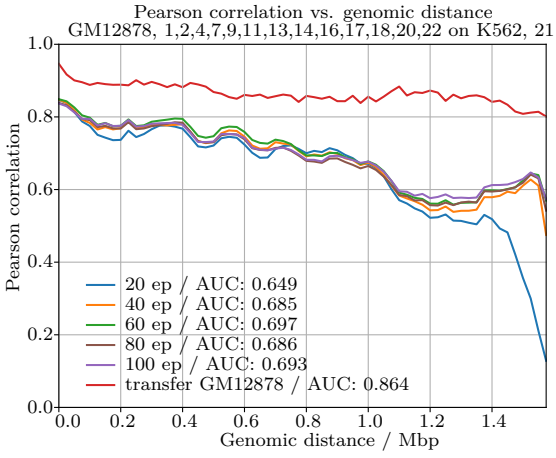
(b) chr5



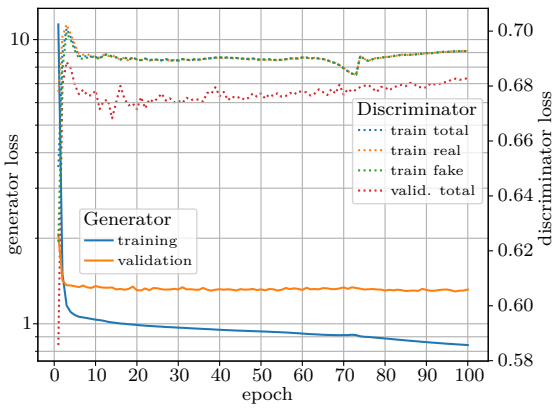
(c) chr10



(d) chr19

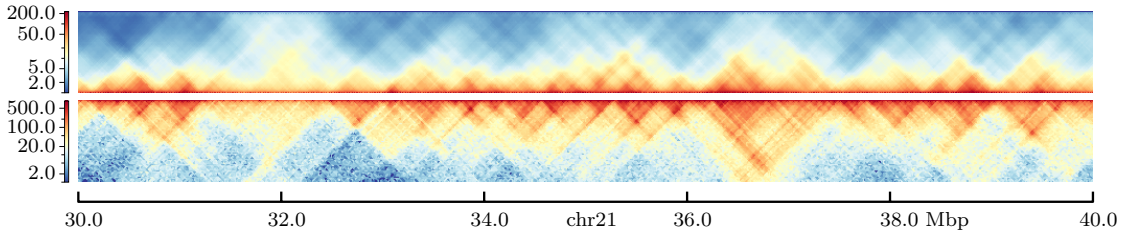


(e) chr21

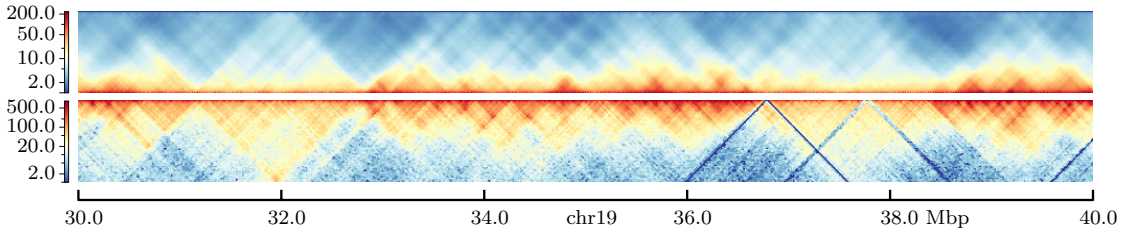


(f) Learning progress

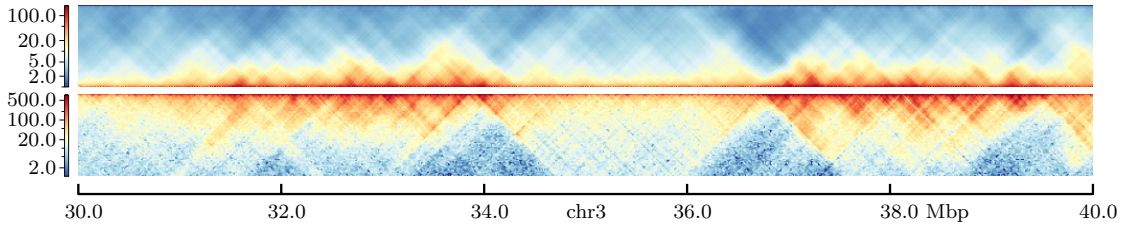
Figure 65: Results / metrics, cGAN, $w = 64$, trained on GM12878 chr14 only; prediction on K562, typical test chromosomes



(a) Example region 1

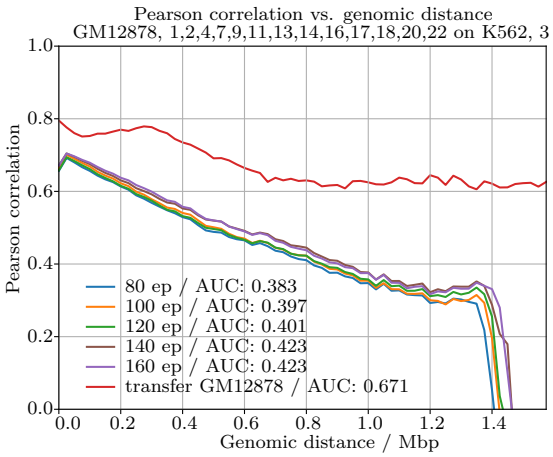


(b) Example region 2

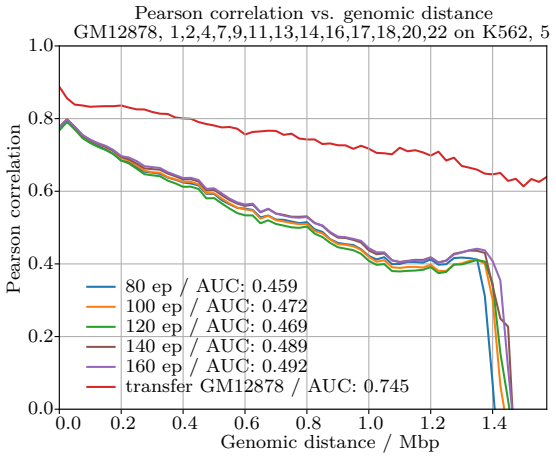


(c) Example region 3

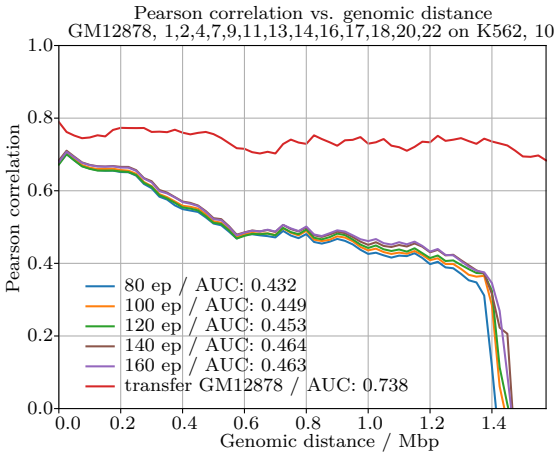
Figure 66: Example predictions, cGAN, $w = 64$, trained on GM12878 chr14 only; prediction on K562, 100 epochs



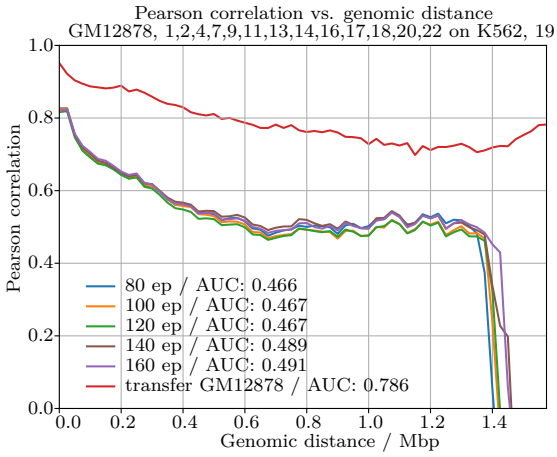
(a) chr3



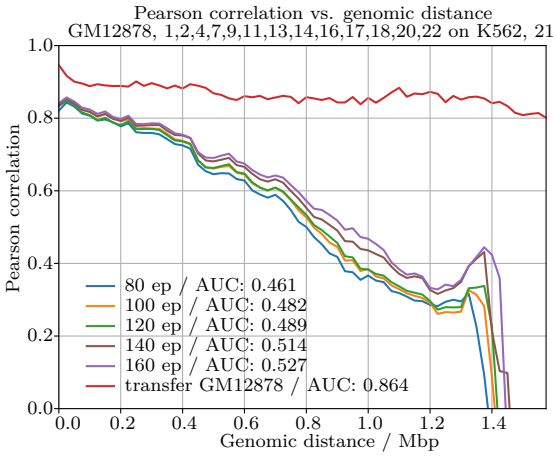
(b) chr5



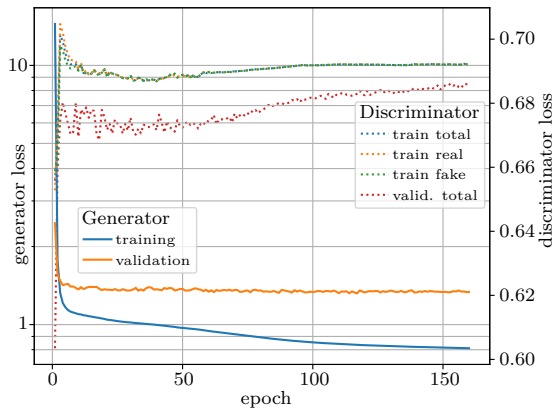
(c) chr10



(d) chr19

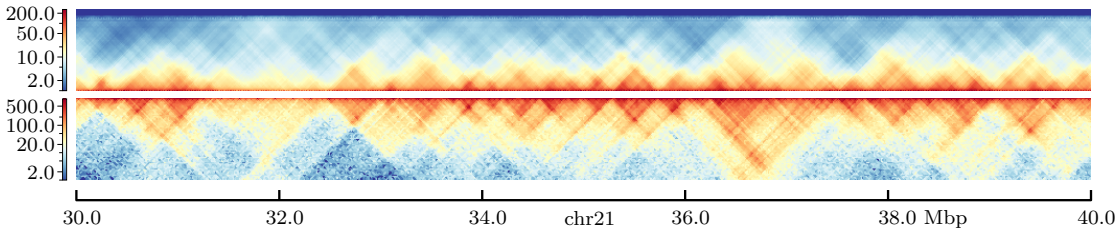


(e) chr21

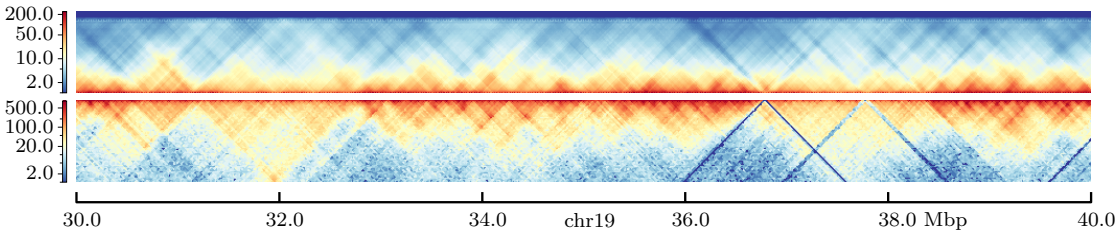


(f) learning progress

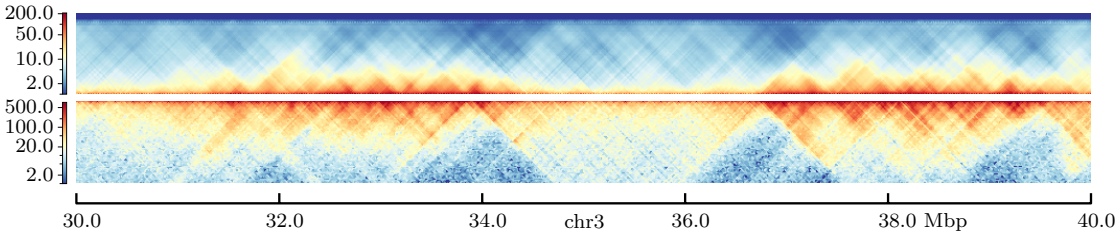
Figure 67: Results / metrics cGAN, $w = 64$, trained on GM12878 chr17 only; prediction on K562, typical test chromosomes



(a) Example region 1



(b) Example region 2



(c) Example region 3

Figure 68: Example predictions cGAN, , $w = 64$, trained on GM12878 chr17 only; prediction on K562, 160 epochs

References

[1] 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.

[2] J. Y. Lee and T. L. Orr-Weaver. "Chromatin". In: *Encyclopedia of Genetics*. Elsevier, 2001, pp. 340–343. DOI: 10.1006/rwgn.2001.0199.

[3] Andrea Smallwood and Bing Ren. "Genome organization and long-range regulation of gene expression by enhancers". In: *Current Opinion in Cell Biology* 25.3 (June 2013), pp. 387–394. DOI: 10.1016/j.ceb.2013.02.005.

[4] David U. Gorkin, Danny Leung and Bing Ren. "The 3D Genome in Transcriptional Regulation and Pluripotency". In: *Cell Stem Cell* 14.6 (June 2014), pp. 762–775. DOI: 10.1016/j.stem.2014.05.017.

[5] Keerthi T. Chathoth and Nicolae Radu Zabet. "Chromatin architecture reorganization during neuronal cell differentiation in Drosophila genome". In: *Genome Research* 29.4 (Feb. 2019), pp. 613–625. DOI: 10.1101/gr.246710.118.

[6] Haoyue Zhang et al. "Chromatin structure dynamics during the mitosis-to-G1 phase transition". In: *Nature* 576.7785 (Nov. 2019), pp. 158–162. DOI: 10.1038/s41586-019-1778-y.

[7] Jennifer E. Phillips and Victor G. Corces. "CTCF: Master Weaver of the Genome". In: *Cell* 137.7 (June 2009), pp. 1194–1211. DOI: 10.1016/j.cell.2009.06.001.

[8] Jennifer E. Phillips-Cremins et al. "Architectural Protein Subclasses Shape 3D Organization of Genomes during Lineage Commitment". In: *Cell* 153.6 (June 2013), pp. 1281–1295. DOI: 10.1016/j.cell.2013.04.053.

[9] Jesse R. Dixon et al. "Chromatin architecture reorganization during stem cell differentiation". In: *Nature* 518.7539 (Feb. 2015), pp. 331–336. DOI: 10.1038/nature14222.

[10] E. Lieberman-Aiden et al. "Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome". In: *Science* 326.5950 (Oct. 2009), pp. 289–293. DOI: 10.1126/science.1181369.

[11] 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 (Dec. 2014), pp. 1665–1680. DOI: 10.1016/j.cell.2014.11.021.

[12] Houda Belaghzal, Job Dekker and Johan H. Gibcus. "Hi-C 2.0: An optimized Hi-C procedure for high-resolution genome-wide mapping of chromosome conformation". In: *Methods* 123 (July 2017), pp. 56–65. DOI: 10.1016/j.ymeth.2017.04.004.

[13] Ralf Krauth. *Improving predictions of Hi-C matrices from ChIP-seq data*. Tech. rep. Albert-Ludwigs Universität Freiburg, 2020. URL: https://github.com/MasterprojectRK/HiCPrediction.

[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] ENCODE Project Consortium. “An integrated encyclopedia of DNA elements in the human genome”. In: *Nature* 489.7414 (Sept. 2012), pp. 57–74. DOI: 10.1038/nature11247. URL: https://www.encodeproject.org.

[17] Carrie A. Davis et al. “The Encyclopedia of DNA elements (ENCODE): data portal update”. In: *Nucleic Acids Research* 46.D1 (Nov. 2017), pp. D794–D801. DOI: 10.1093/nar/gkx1081.

[18] Boyan Bonev and Giacomo Cavalli. “Organization and function of the 3D genome”. In: *Nature Reviews Genetics* 17.11 (Oct. 2016), pp. 661–678. DOI: 10.1038/nrg.2016.112.

[19] A. Tsirikoglou, G. Eilertsen and J. Unger. “A Survey of Image Synthesis Methods for Visual Machine Learning”. In: *Computer Graphics Forum* 39.6 (Sept. 2020), pp. 426–451. DOI: 10.1111/cgf.14047.

[20] Chris A. Brackley et al. “Predicting the three-dimensional folding of cis-regulatory regions in mammalian genomes using bioinformatic data and polymer models”. In: *Genome Biology* 17.1 (Mar. 2016). DOI: 10.1186/s13059-016-0909-0.

[21] Quinn MacPherson, Bruno Beltran and Andrew J. Spakowitz. “Bottom–up modeling of chromatin segregation due to epigenetic modifications”. In: *Proceedings of the National Academy of Sciences* 115.50 (Nov. 2018), pp. 12739–12744. DOI: 10.1073/pnas.1812268115.

[22] 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.

[23] Yifeng Qi and Bin Zhang. “Predicting three-dimensional genome organization with chromatin states”. In: *PLOS Computational Biology* 15.6 (June 2019). Ed. by Jian Ma, e1007024. DOI: 10.1371/journal.pcbi.1007024.

[24] Pau Farré and Eldon Emberly. “A maximum-entropy model for predicting chromatin contacts”. In: *PLOS Computational Biology* 14.2 (Feb. 2018). Ed. by Alexandre V. Morozov, e1005956. DOI: 10.1371/journal.pcbi.1005956.

[25] Jian Zhou and Olga G. Troyanskaya. “Probabilistic modelling of chromatin code landscape reveals functional diversity of enhancer-like chromatin states”. In: *Nature Communications* 7.1 (Feb. 2016). DOI: 10.1038/ncomms10528.

[26] Ziad Al Bkhetan and Dariusz Plewczynski. “Three-dimensional Epigenome Statistical Model: Genome-wide Chromatin Looping Prediction”. In: *Scientific Reports* 8.1 (Mar. 2018). DOI: 10.1038/s41598-018-23276-8.

[27] Yan Kai et al. “Predicting CTCF-mediated chromatin interactions by integrating genomic and epigenomic features”. In: *Nature Communications* 9.1 (Oct. 2018). DOI: 10.1038/s41467-018-06664-6.

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

[29] Laura D. Martens et al. “Identifying regulatory and spatial genomic architectural elements using cell type independent machine and deep learning models”. In: (Apr. 2020). DOI: 10.1101/2020.04.19.049585.

[30] Pau Farré et al. “Dense neural networks for predicting chromatin conformation”. In: *BMC Bioinformatics* 19.1 (Oct. 2018). DOI: 10.1186/s12859-018-2286-z.

[31] Shashank Singh et al. "Predicting enhancer-promoter interaction from genomic sequence with deep neural networks". In: *Quantitative Biology* 7.2 (June 2019), pp. 122–137. DOI: 10.1007/s40484-019-0154-0.

[32] Rui Peng. *Predicting High-order Chromatin Interactions from Human Genomic Sequence using Deep Neural Networks*. Tech. rep. Carnegie Mellon University, 2017. URL: https://www.ml.cmu.edu/research/dap-papers/F17/dap-peng-rui.pdf.

[33] Shashank Singh et al. "Predicting Enhancer-Promoter Interaction from Genomic Sequence with Deep Neural Networks". In: *bioRxiv* (Nov. 2016). DOI: 10.1101/085241.

[34] Jacob Schreiber et al. "Nucleotide sequence and DNaseI sensitivity are predictive of 3D chromatin architecture". In: *bioRxiv* (Jan. 2017). DOI: 10.1101/103614.

[35] Geoff Fudenberg, David R. Kelley and Katherine S. Pollard. "Predicting 3D genome folding from DNA sequence with Akita". In: *Nature Methods* 17.11 (Oct. 2020), pp. 1111–1117. DOI: 10.1038/s41592-020-0958-x.

[36] Geoff Fudenberg, David R. Kelley and Katherine S. Pollard. "Predicting 3D genome folding from DNA sequence". In: *bioRxiv* (Oct. 2019). DOI: 10.1101/800060.

[37] 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.

[38] Sarvesh Nikumbh and Nico Pfeifer. "Genetic sequence-based prediction of long-range chromatin interactions suggests a potential role of short tandem repeat sequences in genome organization". In: *BMC Bioinformatics* 18.1 (Apr. 2017). DOI: 10.1186/s12859-017-1624-x.

[39] Tong Liu and Zheng Wang. "HiCNN2: Enhancing the Resolution of Hi-C Data Using an Ensemble of Convolutional Neural Networks". In: *Genes* 10.11 (Oct. 2019), p. 862. DOI: 10.3390/genes10110862.

[40] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014, pp. 2672–2680. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[41] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[42] Qiao Liu, Hairong Lv and Rui Jiang. "hicGAN infers super resolution Hi-C data with generative adversarial networks". In: *Bioinformatics* 35.14 (July 2019), pp. i99–i107. DOI: 10.1093/bioinformatics/btz317.

[43] Hao Hong et al. "DeepHiC: A generative adversarial network for enhancing Hi-C data resolution". In: *PLOS Computational Biology* 16.2 (Feb. 2020). Ed. by Ferhat Ay, e1007287. DOI: 10.1371/journal.pcbi.1007287.

[44] Michael C. Dimmick, Leo J. Lee and Brendan J. Frey. "HiCSR: a Hi-C super-resolution framework for producing highly realistic contact maps". In: *bioRxiv* (Feb. 2020). DOI: 10.1101/2020.02.24.961714.

[45] Kai Huang, Vadim Backman and Igal Szleifer. "Interphase chromatin as a self-returning random walk: Can DNA fold into liquid trees?" In: (Sept. 2018). DOI: 10.1101/413872.

[46] Artemi Bendandi et al. “Chromatin Compaction Multiscale Modeling: A Complex Synergy Between Theory, Simulation, and Experiment”. In: *Frontiers in Molecular Biosciences* 7 (Feb. 2020). DOI: 10.3389/fmolb.2020.00015.

[47] Andre Bajorat. *Hi-C Predictions based on protein levels*. Tech. rep. Albert-Ludwigs Universität Freiburg, 2019. URL: https://www.bioinf.uni-freiburg.de/Lehre/Theses/TP_Andre_Bajorat.pdf.

[48] Bernd Schuettengruber et al. “Cooperativity, Specificity, and Evolutionary Stability of Polycomb Targeting in Drosophila”. In: *Cell Reports* 9.1 (Oct. 2014), pp. 219–233. DOI: 10.1016/j.celrep.2014.08.072.

[49] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017, pp. 5967–5976. DOI: 10.1109/CVPR.2017.632.

[50] Yingjing Lu. “The Level Weighted Structural Similarity Loss: A Step Away from the MSE”. In: (Apr. 30, 2019). arXiv: 1904.13362 [cs.CV].

[51] Z. Wang, E. P. Simoncelli and A. C. Bovik. “Multiscale structural similarity for image quality assessment”. In: *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*. IEEE. DOI: 10.1109/acssc.2003.1292216.

[52] Hang Zhao et al. “Loss Functions for Image Restoration With Neural Networks”. In: *IEEE Transactions on Computational Imaging* 3.1 (Mar. 2017), pp. 47–57. DOI: 10.1109/tci.2016.2644865.

[53] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1409.1556.

[54] Justin Johnson, Alexandre Alahi and Li Fei-Fei. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”. In: *Computer Vision – ECCV 2016*. Springer International Publishing, Mar. 27, 2016, pp. 694–711. DOI: 10.1007/978-3-319-46475-6_43.

[55] Leonid I. Rudin, Stanley Osher and Emad Fatemi. “Nonlinear total variation based noise removal algorithms”. In: *Physica D: Nonlinear Phenomena* 60.1-4 (Nov. 1992), pp. 259–268. DOI: 10.1016/0167-2789(92)90242-f.

[56] Rola Dali and Mathieu Blanchette. “A critical assessment of topologically associating domain prediction tools”. In: *Nucleic Acids Research* 45.6 (Mar. 2017), pp. 2994–3005. DOI: 10.1093/nar/gkx145.

[57] Marie Zufferey et al. “Comparison of computational methods for the identification of topologically associating domains”. In: *Genome Biology* 19.1 (Dec. 2018). DOI: 10.1186/s13059-018-1596-9.

[58] Emily Crane et al. “Condensin-driven remodelling of X chromosome topology during dosage compensation”. In: *Nature* 523.7559 (June 2015), pp. 240–244. DOI: 10.1038/nature14450.

[59] Hanjun Shin et al. “TopDom: an efficient and deterministic method for identifying topological domains in genomes”. In: *Nucleic Acids Research* 44.7 (Dec. 2015), e70–e70. DOI: 10.1093/nar/gkv1505.

[60] Joachim Wolff et al. "Galaxy HiCExplorer: a web server for reproducible Hi-C data analysis, quality control and visualization". In: *Nucleic Acids Research* 46.W1 (June 2018), W11–W16. DOI: 10.1093/nar/gky504.

[61] Lei Wang et al. "A State-of-the-Art Review on Image Synthesis with Generative Adversarial Networks". In: *IEEE Access* 8 (2020), pp. 63514–63537. DOI: 10.1109/access.2020.2982224.

[62] Scott Reed et al. "Generative Adversarial Text to Image Synthesis". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. JMLR.org, 2016, pp. 1060–1069.

[63] Han Zhang et al. "StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.8 (Aug. 2019), pp. 1947–1962. DOI: 10.1109/tpami.2018.2856256.

[64] Minfeng Zhu et al. "DM-GAN: Dynamic Memory Generative Adversarial Networks for Text-To-Image Synthesis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019, pp. 5802–5810.

[65] M. Tao et al. "DF-GAN: Deep Fusion Generative Adversarial Networks for Text-to-Image Synthesis". In: *ArXiv* abs/2008.05865 (2020).

[66] Olaf Ronneberger, Philipp Fischer and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Lecture Notes in Computer Science*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28.

[67] Scott Reed et al. "Learning Deep Representations of Fine-Grained Visual Descriptions". In: *IEEE Computer Vision and Pattern Recognition (CVPR)*. 2016.

[68] Tao Xu et al. "AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018. DOI: 10.1109/cvpr.2018.00143.

[69] Andrew L. Maas, Awni Y. Hannun and Andrew Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[70] 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.

[71] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016. DOI: 10.1109/cvpr.2016.308.

[72] Lucille Lopez-Delisle et al. "pyGenomeTracks: reproducible plots for multivariate genomic datasets". In: *Bioinformatics* (Aug. 2020). Ed. by Robinson Peter. DOI: 10.1093/bioinformatics/btaa692.

[73] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[74] François Chollet et al. *Keras*. 2015. URL: https://keras.io.

[75] Tensorflow authors. *tf.image.total_variation*. Online. URL: https://www.tensorflow.org/api_docs/python/tf/image/total_variation (visited on 02/25/2021).

[76] Tensorflow authors. *Pix2Pix Tutorial*. Online. URL: https://www.tensorflow.org/tutorials/generative/pix2pix (visited on 12/01/2020).

[77] Jason Brownlee. *How to Implement Pix2Pix GAN Models From Scratch With Keras*. Online. URL: https://machinelearningmastery.com/how-to-implement-pix2pix-gan-models-from-scratch-with-keras/ (visited on 12/01/2020).

[78] and S. Roy et al. "Identification of Functional Elements and Regulatory Circuits by Drosophila modENCODE". In: *Science* 330.6012 (Dec. 2010), pp. 1787–1797. DOI: 10.1126/science.1198374.

[79] Ralf Krauth. *Hi-cGAN github repository*. Online. URL: https://github.com/MasterprojectRK/Hi-cGAN.

[80] Ralf Krauth. *DNN-pCC github repository*. Online. URL: https://github.com/MasterprojectRK/DNN-pCC.

Acronyms

AUC area under the correlation curve.

cGAN conditional generative adversarial network.

ChIA-PET chromatin interaction analysis by paired-end tag sequencing.

ChIP-seq chromatin immunoprecipitation followed by sequencing.

CNN convolutional neural network.

DamID DNA adenine methyltransferase identification.

DNN dense neural network.

GAN generative adversarial network.

LSTM long-short-term memory.

TAD topologically associating domain.