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 23. Februar 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 existing 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 genomes. 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.

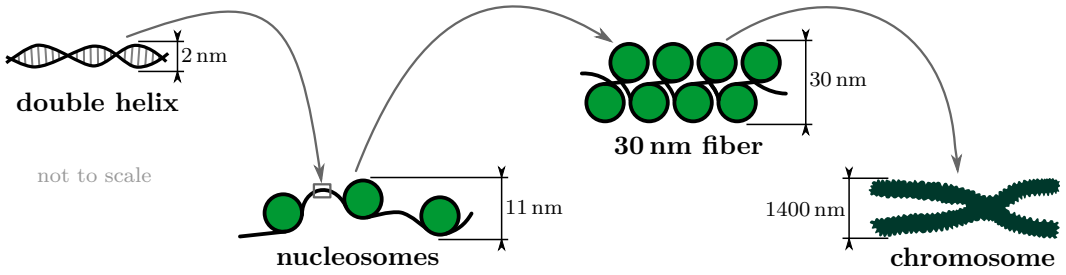


Figure 1: spatial chromatin structures (simplified, cf. [2])

While the spatial structure of chromatin outlined above brings along compaction (fig. 1 from left to right), e. g. to make chromosomes fit into eucaryotic cell nuclei, 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 gene enhancers and the relevant promoters [3, 4]. Since enhancers can be up to a million basepairs away from the promoters, the two can only interact by means of spatial DNA structure. Many 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, certain proteins like CTCF or modified histones are already well known to mediate or prevent DNA-DNA contacts [7, 8, 9].

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.

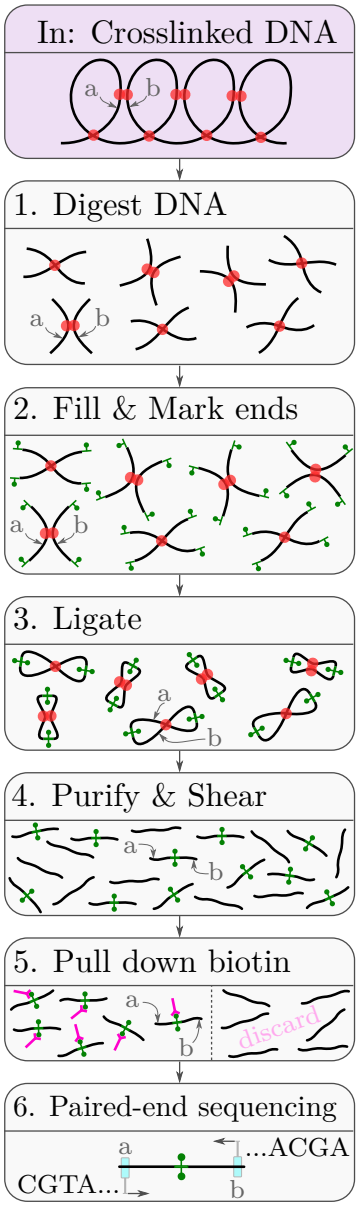


Figure 2: Hi-C lab process

The typical input (In) to Hi-C consists of several millions of cells, which are treated chemically to fix existing DNA-DNA contacts, commoly 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 just 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).

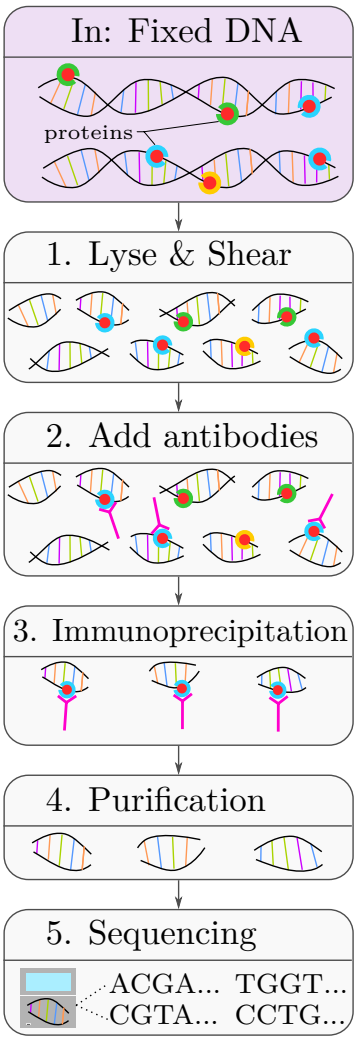


Figure 3: ChIP-seq lab process

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 be 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 aligned reads from the ChIP-seq experiment have been used directly, 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 conducted only by few labs and for comparatively few organisms and cell lines so far. For example, as of January 2021, Hi-C assays are available for just 26 of more than 150 human cell lines listed in the Encyclopedia of DNA elements [16, 17], and these have been produced by only five university labs.

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 would be very helpful.

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)

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 of 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 postprocessing.

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 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 with this approach, highly interacting regions are still often well identifiable.

Three further approaches in the field make use 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 two genomic "windows", using ChIP-seq data of 13 chromatin factors and the genomic distance of the two windows. The published results for five human cell lines look interesting.

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. However, 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 from 50 chromatin factors into a one-dimensional chromatin vector, which is then processed by a dense neural network (DNN). This allows predicting 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 it 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. initally 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 are 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, deep convolutional neural networks – modified from established image super-resolution networks XXX– have been used 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 (GANs), 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 GAN setup with two competitive networks – a generator, which is trained to produce realistic high-resolution Hi-C matrices from its low-resolution inputs, and a discriminator, which tries to distinguish real Hi-C matrices from generated ones and thus serves as a "critique", an additional loss function, for the generator. 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 used and the activation functions applied within the blocks. Furthermore, Dimmick et al. and Hong et al. include additional loss functions beyond standard generator- and adversarial losses. This includes perceptual losses derived from other (pre-trained) neural networks to obtain "visually good" Hi-C matrices and total variation loss to suppress noise while maintaining edges. 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 develoment.

The chromatin-modeling based approaches [20, 21, 22, 23] allow indirect derivation of Hi-C matrices by performing sufficient simulation passes and estimating contacts from the resulting model ensembles. Depending on the chosen chromatin model – 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 not all constraints for chromatin modeling are known or cannot be considered in the models 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 there seems no obvious correlation between sequence and structure – instead, 3D conformation can vary considerably for different cell lines of the same organism, which all share the same DNA. 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 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. However, this approach has extensively been investigated in two previous study projects at the university of Freiburg, and the results could not be reproduced for unknown reasons [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.

3.1 Dense Neural Network approach

In their 2008 paper [30], Pau Farré, Alexandre Heurtau, Olivier Cuvier and Eldon Emberly propose 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. This thesis tries to build on the success of their method by extending the comparatively simple neural network in various ways, modifying the binsizes of the Hi-C matrices and changing the learning process. 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 simplied form in fig. 4, see section 4.3.1 for technical details.

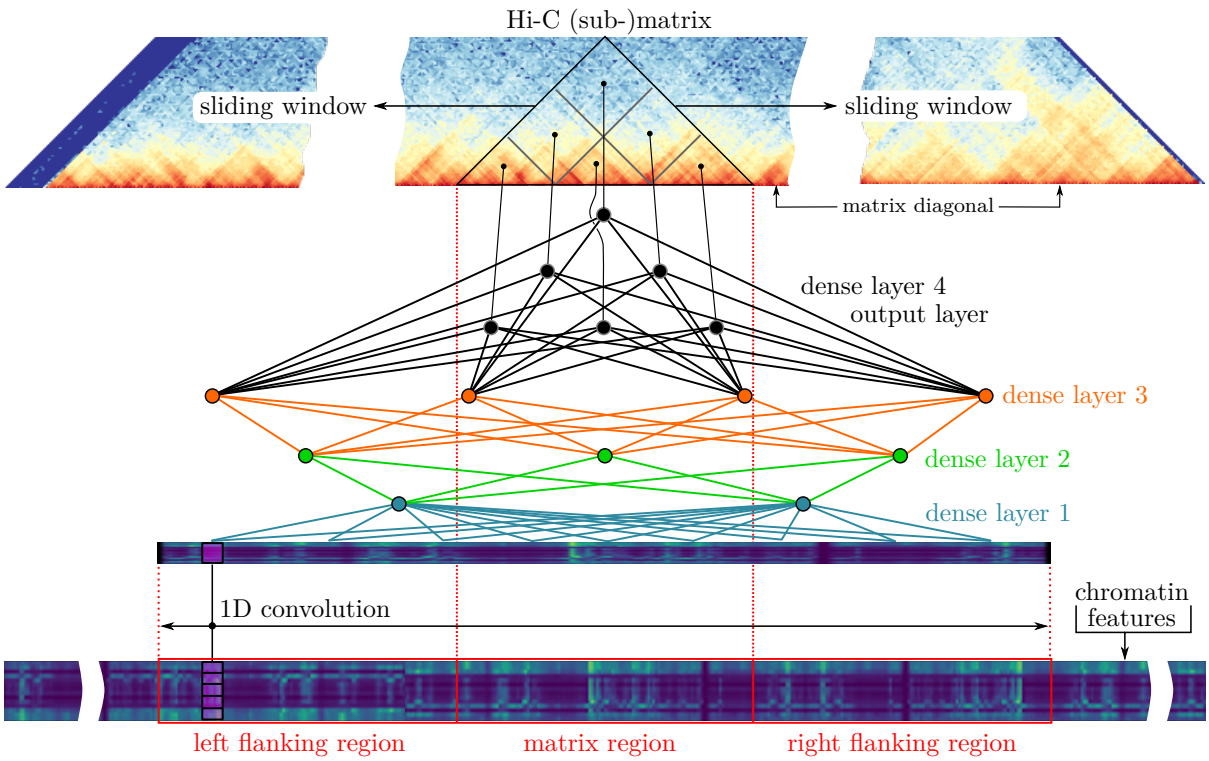


Figure 4: Principle of basic dense neural network

Since the network implements a supervised learning technique, it requires two kinds of inputs for training – ChIP-seq data of the chosen chromatin factors and target Hi-C matrices for each training chromosome. However, it is generally infeasible to learn Hi-C matrices for complete chromosomes at once with machine learning approaches, because the latter heavily depend on availability of training data. The target matrices are thus taken as submatrices of size $w \times w$ with fixed windowsize 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 $3w \times n$ 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. Training samples are then obtained from the input data by sliding the input windows along the diagonal of the target Hi-C matrix. For the technical details of the sample generation process, see section 4.1.3.

Within the network, a 1D convolutional filter compresses the $3w \times n$ input arrays to 1D vectors of size $3w \times 1$, 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 4.3.1.

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 4.3.1.

Farré et al. propose a windowsize of $w = 80$ at $b_{feat} = b_{mat} = 10\,\text{kbp}$. However, larger binsizes of $b_{feat} = b_{mat} = 25\,\text{kbp}$ were found beneficial for most of the data used throughout this master thesis. Additionally, larger binsizes allow for a higher coverage of the target matrix at the same windowsize, because the windowsiz is specified in bins, and obviously $10\,w < 25\,w$. Results for both binsize 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 might 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 4.3.2. 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.

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 is 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 binsize $b = 25\,\text{kbp}$, 4 bins already correspond to 100 kbp. 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.

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]. ChIP-seq experiments can usually be binned at smaller binsizes than Hi-C data due to the nature of the process. 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 4.1.5 for the technical details. This yields an input data size of $k \cdot 3w \times n$, which is then again compressed to a $3w \times 1$ vector by a 1D convolutional filter with kernel size n and strides k. For practical reasons, $k = 5$ was chosen for the thesis at hand, and the results for binsizes $b_{mat} = 25\,\text{kbp}$, $b_{feat} = 5\,\text{kbp}$ are shown in XXX.

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

In image regression tasks, optimizing for mean squared error is known to produce blurry images, because it is computed independently for each image pixel, ignoring spatial proximity [48, 49]. Thus, another approach for improving the predictions of the basic neural network is to use a different loss function.

It has been shown that loss functions based on the (multiscale-)structural similarity index (SSIM) [50]. 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 [51], Lu proposed a custom level-weighted SSIM loss [49]. The results were better than with L1- or L2 loss alone, but sometimes not much – depending on the machine learning model in use.

Another type of loss function used in image processing is the so-called perceptual- or perception loss. The idea here is not to compute L1 or L2 loss directly from the output of the network to be trained, but instead use a pre-trained loss network to determine structures in "predicted" and "real" images and then compute e. g. L1 or L2 loss on these structures, fig. 5. The optimization

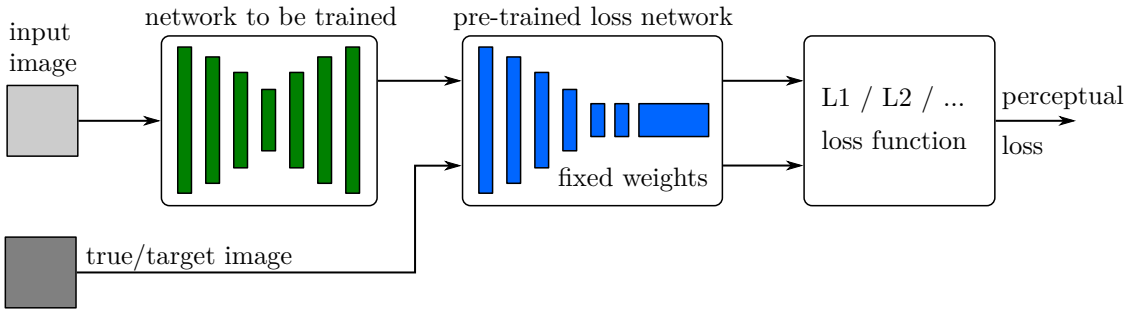


Figure 5: perceptual loss

of the trainable weights can be performed as usual, for example with gradient descent and backpropagation, simply keeping the weights of the loss networks constant. Often, complex image classification networks like VGG-16 [52] are taken as loss network, e. g. in the well-known style-transfer network by Johnson et al. [53], because these are known to be good at detecting relevant structures in images.

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 VGG-16 and total variation loss L_{TV} to reduce noise in the output while preserving edges [54]. 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 4.3.3 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 $tad(z_{pred})$, where z_{pred} is a predicted submatrix, this might be improved by directly optimizing a loss function as shown in eq. (2).

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

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 [55, 56]. 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 [56]. 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 [57] was developed. Figure 6 exemplarily shows its basic idea for a 16×16 submatrix with windowsize 4. First, the mean is taken from diamond-shaped (or rhombus-shaped) matrix cutouts along the 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

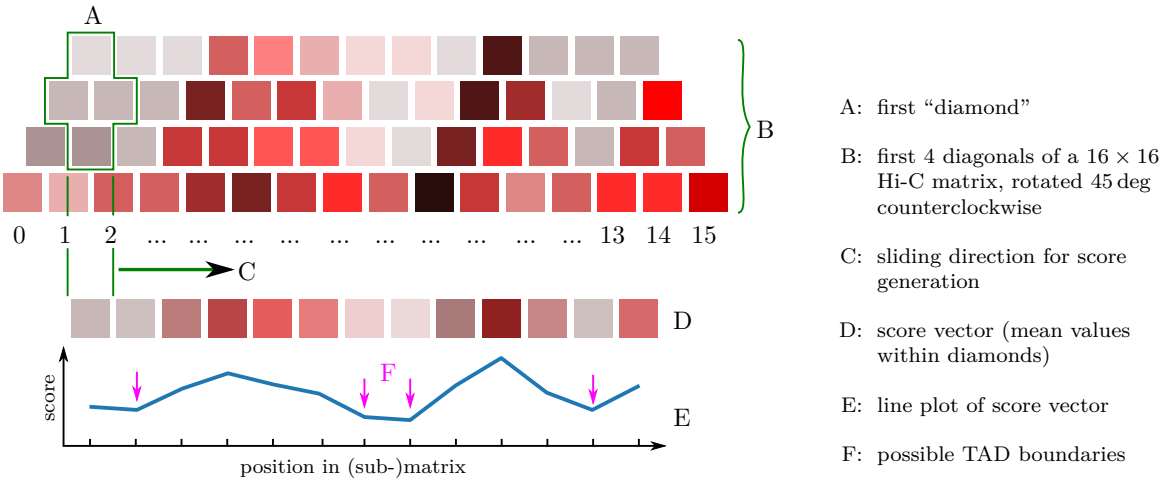


Figure 6: score vector generation for TAD-based loss

[58] and hicFindTADs [59, 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, windowsize $w = 80$ and diamond size $ds = 12$ were used at binsizes of 25 kbp. For technical details please refer to section 4.3.4.

3.1.5 Modifying binsize and windowsize

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 binsizes $b_{feat} = b_{mat} = 50\,\text{kbp}$, keeping the windowsize 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 binsizes. 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-)binsizes 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 windowsizes 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 windowsize *in bins* and the relation $k = b_{mat}/b_{feat}$ between the binsizes 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 binsizes.

- use trapezoids, i.e. capped larger submatrices and flankingsize smaller than windowsize
- rationale: larger windowsize 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 [60]. Among their strenghts is image synthesis from textual descriptions [61, 62, 63, 64] – 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 [48].

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

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

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 [48], 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

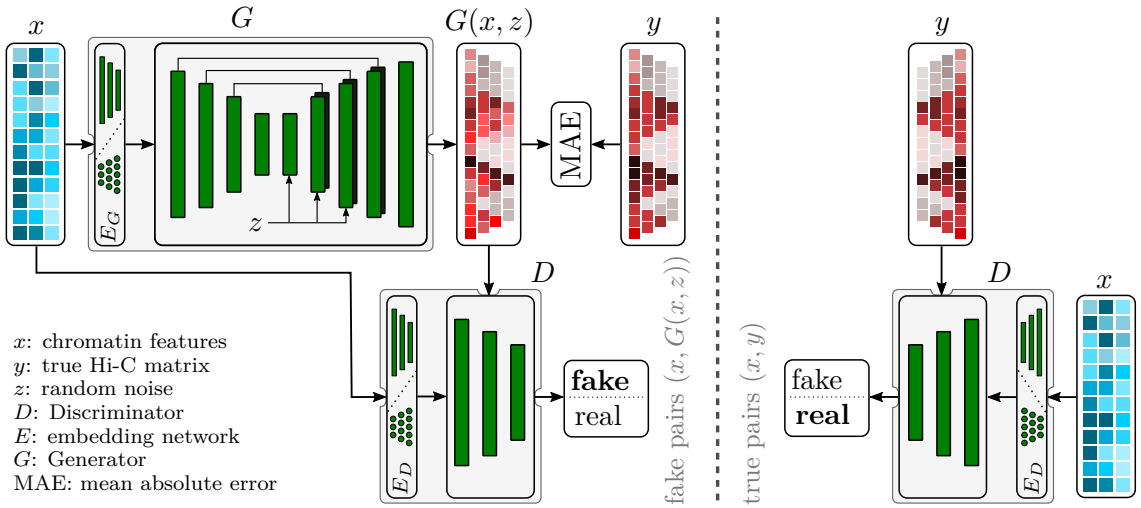


Figure 7: cGAN approach

architecture due to Ronneberger, Fischer and Brox [65], while the discriminator is implemented as a patchGAN-discriminator developed by Isola et al. especially for pix2pix [48]. 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 at the appropriate places, see section 4.4.1 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 4.4.1 for details.

For sample generation, we relied on the same method as described for the dense neural network described above with a few minor adaptations, see section 4.1.4. 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 approaches were tried, 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 [66, 67], 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, we therefore relied on the extended loss function proposed by Isola et al. [48], amended by a TV loss function L_{TV} for noise reduction, eq. (5).

$$L_{total} = \lambda_{cGAN}\, L_{cGAN}(G, D) + \lambda_{MAE} L_{MAE}(G) + \lambda_{TV} L_{TV}(G) \tag{5}$$

Here, L_{MAE} is the mean absolute error (L1 error) between the generated output $G(x, z)$ and the true Hi-C submatrices y (as grayscale images), and l_{TV} is a total variation loss as in eq. (1) and eq. (8). The original idea behind using mean absolute error and PatchGAN in combination is

that optimizing for MAE will minimize low-frequency errors, while optimizing the PatchGAN-loss with its small patches helps reducing high-frequency errors [48]. 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.

Finally, the loss function was optimized with the Adam optimizer, alternately training discriminator and generator, see section 4.4.1 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 [48]. 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 towards pix2pix, and U-Net architectures in general, is that they can achieve good performance even with comparatively few training samples [48, 65].

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 $(w, w, 1)$. 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. [62], albeit much less sophisticated. Naturally, this approach also allows to pre-train the dense network as shown above and then use transfer learning to provide better starting values for 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 windowsize w. Taken together with the three fully connected layers underneath, this leads to a superlinear increase in the number of parameters along w, further aggravated by the fact that the cGAN requires w to be powers of two, cf. section 4.4.1 and 4.4.2 (table 4, p. 34). The DNN embedding has thus only been explored for $w = 64$, both with and without pretraining. The results are to be found in section 5.2.1.

3.2.3 Using a CNN for feature embedding

Since the results from the DNN embedding were not convincing, another approach based on convolutional layers was created. Here, the idea was to have a trainable embedding that keeps the localization information in the chromatin feature data, has less parameters than a dense embedding and can efficiently be trained along with the rest of the network, allowing for a standalone solution.

Since no example for such an embedding was found in literature, a first try was made with a simple 8-layer convolutional network. The complete setup is shown in section 4.4.3 (fig. 18, p. 40). Compared to the dense approach, the number of parameters in the CNN embedding is

widely independent of the windowsize and stays between 4.24 million and 4.29 million for the tested windowsizes 64, 128 and 256, cf. table 4 (p. 34).

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 4.3 and 4.4 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. Finally, the hardware used for the computations within this thesis will be outlined in section 4.5.

4.1 Input data

For the thesis at hand, data from human cell lines GM12878, K562, HMEC, HUVEC and NHEK was used. The exact data sources and data preprocessing for the Hi-C matrices and ChIP-seq data will be outlined in the following subsections 4.1.1 and 4.1.2.

4.1.1 Hi-C matrices

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 a 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].

4.1.2 ChIP-seq data

For this thesis, 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]; it contains structural proteins like CTCF and Cohesin subcomponents RAD21 and SMC3 as well as active/passive markers.

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

4.1.3 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 then stacked into a $l_{feat} \times n$ array. Here, the number of chromatin features was constant for all investigations within this thesis, $n = 14$ (cf. section 4.1.2).

Separate Hi-C matrices for each chromosome were derived from the cooler format as $(l_{mat} \times l_{mat})$-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 now applied to generate training samples for the neural networks G described below. Here, subarrays of size $(3w_{mat} \times n)$ 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 (16×16)-matrix 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.

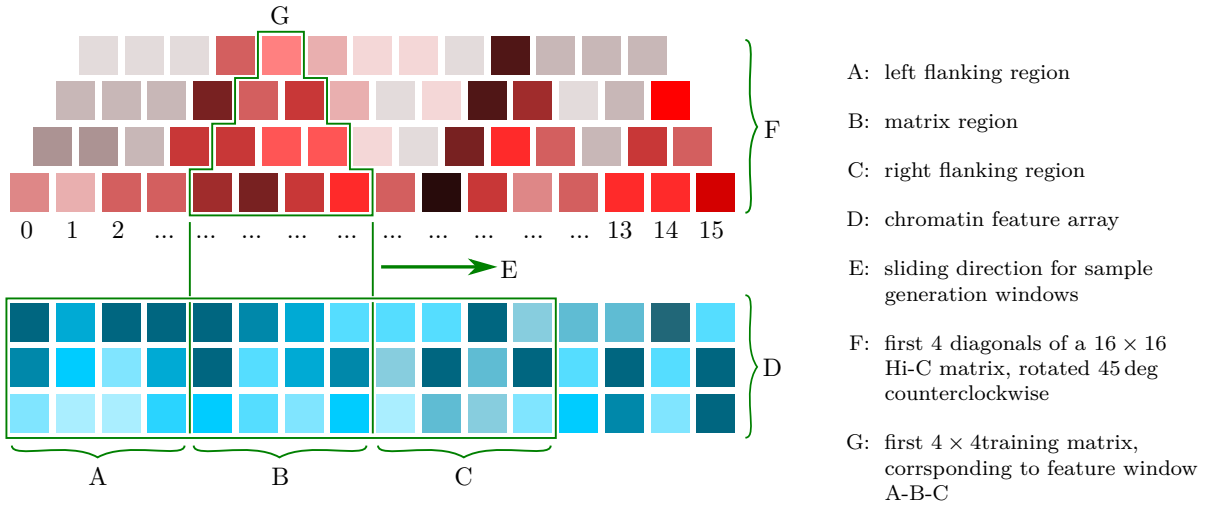


Figure 8: Sample generation process

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 16×16 matrix. 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.

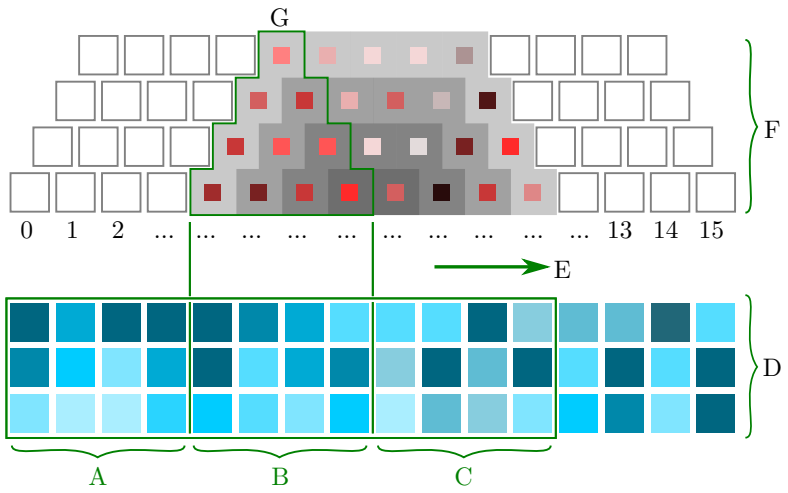
Training samples were drawn from GM12878, chromosomes 1, 2, 4, 7, 9, 11, 13, 14, 16, 17, 18, 20 and 22 of GM12878, validation samples from GM12878, chromosomes 6, 8, 12 and 15 and test samples from K562, chromosomes 3, 5, 10, 19 and 21. This approximately implements a 60:20:20 train:validation:test split, see table 3 for details. Note that windowsize $w = 80$ is 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 4.3.1.

4.1.4 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 4.1.3, with two exceptions. First, Hi-C (sub-)matrices were not entered as vectors corresponding to their upper triangular part, but were instead taken as $w \times w \times 1$ 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

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



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

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.

4.1.5 Generalization of feature binning

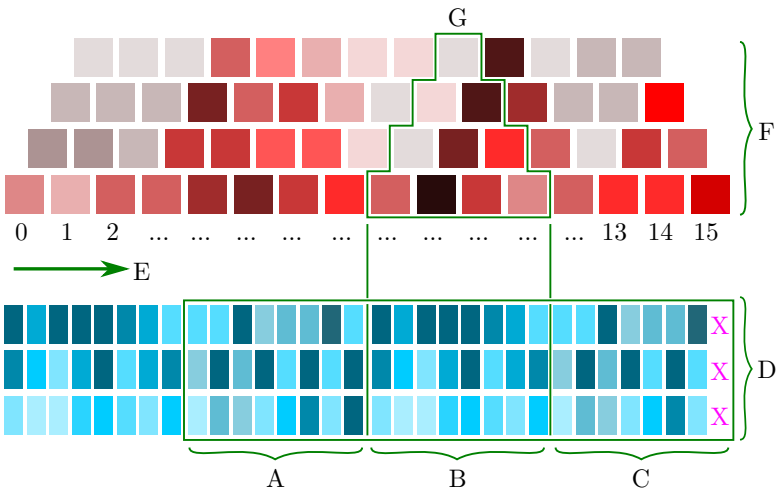
Most of the binning- and sample generation procedures described above also work for binsize relations $k = \frac{b_{mat}}{b_{feat}} \in \mathbb{N}^{>1}$.

The training matrices remain unchanged, i. e. $(l \times l)$-arrays, from which training submatrices of size $w_{mat} \times w_{mat}$ 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 number of bins along the matrix diagonal is generally not k times the number of bins in the feature array, see eq. (6).

$$l_{feat} = \left\lceil \frac{cs}{b_{feat}} \right\rceil = \left\lceil \frac{cs}{k \cdot b_{mat}} \right\rceil \neq k \cdot \left\lceil \frac{cs}{b_{mat}} \right\rceil = k \cdot l_{mat} \tag{6}$$

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



A: left flanking region

B: matrix region

C: right flanking region

D: chromatin features array ($k = 2$)

E: sliding direction for sample generation windows

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

G: last 4×4 training matrix, corrsponding to feature window A-B-C; to be dropped/padded

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

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.

4.2 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. [60, 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 [68], 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 remain zero in the predictions due to the chosen sample generation approach, cf. section 4.1.3, fig. 9 and 4.1.4. Next, the remaining values were grouped according to their genomic distance, up to the maximum distance w, and then Pearson correlations were computed as usual, separately for each group. Additionally, 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. 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 relevant GM12878 matrix as a "prediction" for a given target matrix.

Since relying on one metric alone seemed problematic, pygenometracks [69] was used to plot selected areas of the test set where the characteristics of the results seemed particularly distinct.

In computer graphics, other metrics like the structural similarity index (SSIM), peak signal-to-noise ratio (PSNR) or, especially for images generated by GANs, Frechet inception score (FIS) are more common. However, all of these operate on images, and no useful images seem available

for comparisons within this thesis. Images of complete Hi-C matrices, i. e. for full chromosomes, can be generated from cooler matrices, but contain only zeros outside the first w diagonals. This would lead to false results when compared to the true matrices, whether they are truncated at w or not. Images of $(w \times w)$ sub-matrices, on the other hand, are generated natively within this thesis, but are overlapping and thus highly correlated among each other due to the sliding window approach for sample generation, cf. section 4.1.3 and 4.1.4. Again, the result would be not very meaningful, or at least difficult to interpret.

4.3 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.

4.3.1 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 $b_{feat} = b_{mat} = 10\,\text{kbp}$ 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 the dropout layers following all but the last dense layer 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 $M_s = G_s(\omega)$ and the training submatrices T_s becomes minimal for all N training samples $s \in (1, 2, \ldots, N)$. Here, 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 $(w \times w)$ Hi-C matrix. Formally, one needs to optimize

$$\omega^* = arg\,min_\omega L_2(\omega) = arg\,min_\omega \frac{1}{N} \sum_{s=0}^{N} (M_s - T_s)^2 \tag{7}$$

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 [70, 71], see repository XXX.

4.3.2 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 now 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 $3w \cdot k \times n = 3 \cdot 80 \cdot 5 \times 14 = 1200 \times 14$. Replacing the first convolutional layer of the basic network by a 1D convolutional filter with kernel size $k = 5$ and strides $k = 5$, this input was again compressed to a $3w \times 1 = 240 \times 1$ 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.

4.3.3 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 $w \times w \times 1$ tensors, where w is again the windowsize.

For an image, total variation can be defined as the sum of the absolute differences for neighboring pixel-values in an image, eq. (8) [54]

$$\sum_{i,j} \sqrt{|y_{i+1,j} - y_{i,j}|^2 + |y_{i,j+1} - y_{i,j}|^2} \tag{8}$$

For simplicity the tensorflow implementation from tf.image.total_variation was used, taking the XXXsum XXXmean across batches, as recommended in the tensorflow documentation.

For perception loss, the predicted images and the true images were first fed through a pre-trained VGG-16 network with fixed weights, truncated at layer XXX. The loss was then computed as mean squared error between the "predicted" and "true" output activations of the truncated VGG-16 network.

Let $M_s = G_s(\omega)$ again be the output of the neural network G described above, and T_s the true matrices for training samples s in vector form, and let M'_s and T'_s be their grayscale image counterparts as described above. Furthermore, let $tv(x)$ be the total variation of image x and

$vgg(x)$ the output of the perception loss network for image x. The goal of the modified network was now to find weights w^* such that

$$\omega^* = arg\,min_\omega(\lambda_{MSE}\frac{1}{N}\sum_{s=0}^{N}(M_s - T_s)^2 + \lambda_{TV}\sum_{s=0}^{N} tv(M'_s) + \lambda_{VGG}\frac{1}{N}\sum_{s=0}^{N}(vgg(M'_s) - vgg(T'_s))^2) \quad (9)$$

Weight initialization for network G and minibatching was done as described in section 4.3.1. The weights for the VGG16 network were taken from the pre-trained keras implementation and can here be considered as constants. In addition, 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 $G(\omega)$ could in principle be any of the variants described above in section 4.3.2, but for the thesis at hand, only the initial network from section 4.3.1 was used.

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

Formally, the the following optimization task can be defined by means of a combination between MSE and TAD-score based loss:

$$\omega^* = arg\,min_\omega(\lambda_{MSE}\frac{1}{N}\sum_{s=0}^{N}(M_s - T_s)^2 + \lambda_{score}\frac{1}{N}\sum_{s=0}^{N}(score(M'_s, ds) - score(T'_s, ds))^2 \quad (10)$$

where M_s is again the Hi-C submatrix predicted by the network $G(\omega)$ for sample s and 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 M and T in vector form (upper triangular part) were first converted back to complete, symmetric Hi-C matrices M', 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. (10).

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.

4.4 Hi-cGAN approach

4.4.1 Modified pix2pix network

Several implementations of the original pix2pix network are publicly available, usually for the original image size of 256×256. For the thesis at hand, implementation concepts from two tutorials [72, 73] 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 64. Furthermore, symmetry of the output was enforced by adding its 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 activation of the output layer was changed from tanh to sigmoid, because better results were observed in conjunction with the given Hi-C matrices, probably because these contain no negative values by definition.

Compared to the original pix2pix setup, one downsampling layer was omitted in the discriminator for windowsize $w = \{256, 128\}$ and another one for $w = 64$. This made the discriminator patches larger, especially for the smaller windowsizes, cf. fig. 14. 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 up- and downsampling layers.

Both the discriminator and the generator feature 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 4.4.2 and section 4.4.3.

The loss function was implemented as shown in eq. (5) with parameters $\lambda_{cGAN} = 10^{-5}, \lambda_{MAE} = 1.0, \lambda_{TV} = 10^{-12}$. Optimization was performed on minibatches of size 32,4 and 2 for windowsizes 64, 128 and 256, respectively, using the Adam optimizer with learning rate $2 \cdot 10^{-5}$ 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 4.5.

a) the number of trainable parameters

4.4.2 Using a DNN for 1D-2D embedding

In order to use the DNN described in section 4.3.1 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 simply 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. 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

4.4.3 Using a CNN for 1D-2D 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.

For the three windowsizes $w = \{64, 128, 256\}$ used within this thesis, the embedding network shown in fig. 18 contained about 4.2 to 4.3 million trainable weights, see table 4 for details.

4.5 Hardware

For the thesis, three virtual machines were used to train the neural networks, see table 5. All training for section 5.2 was done on machine 2, 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 5) 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 80 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.

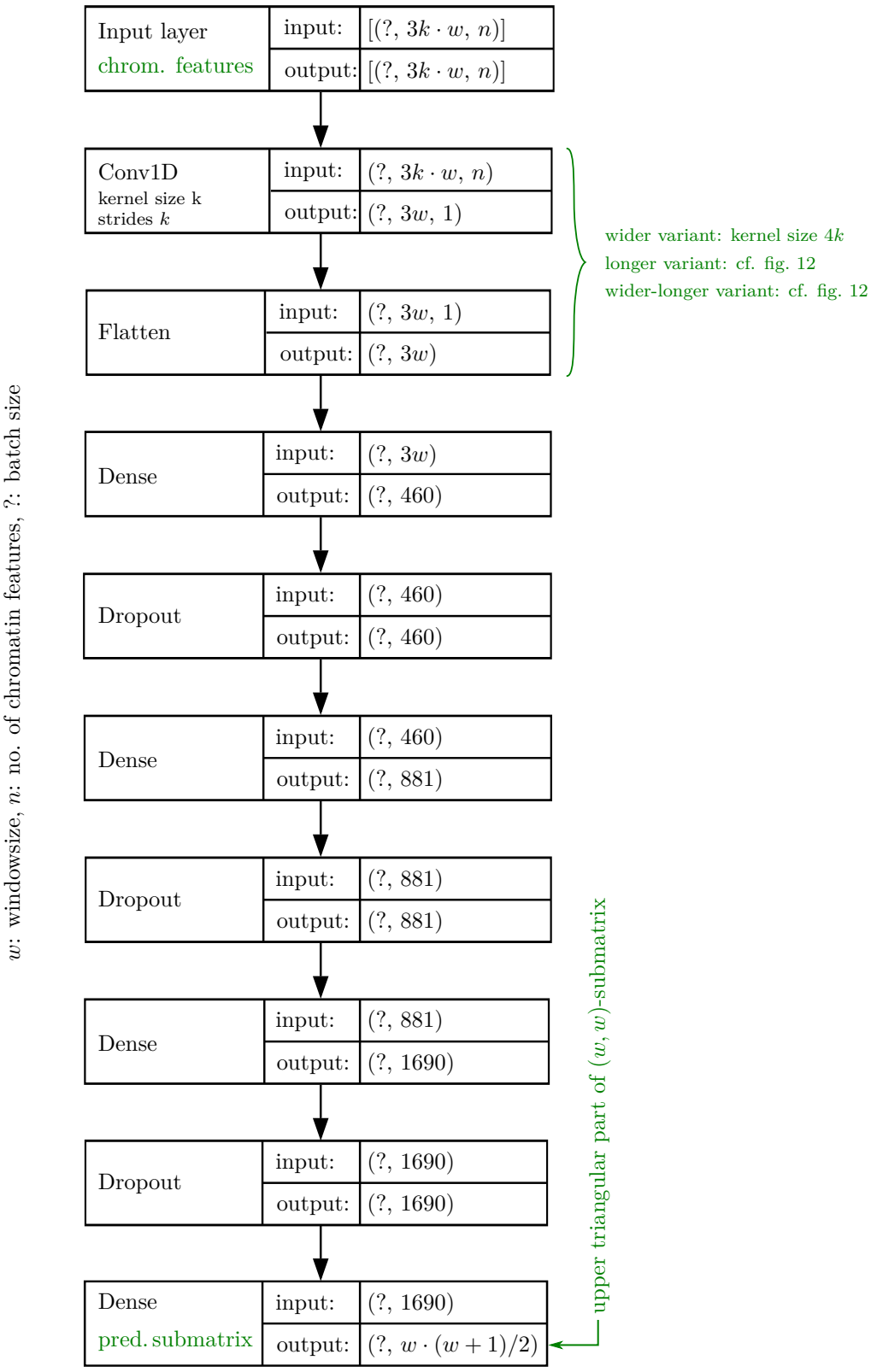


Figure 11: Basic dense neural network with generalized feature binning

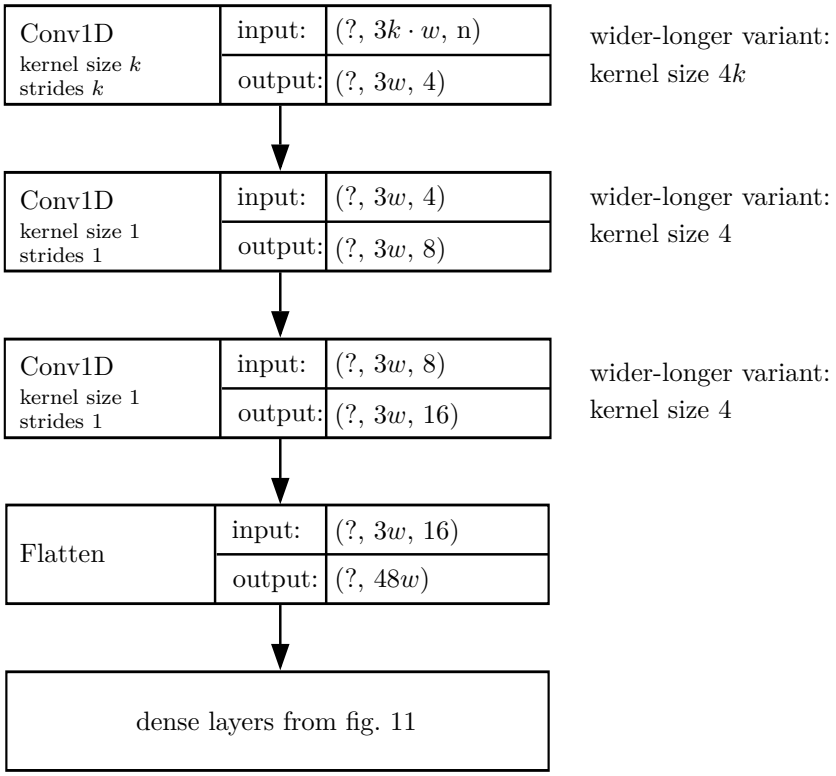


Figure 12: “longer” and “wider-longer” variants for DNN

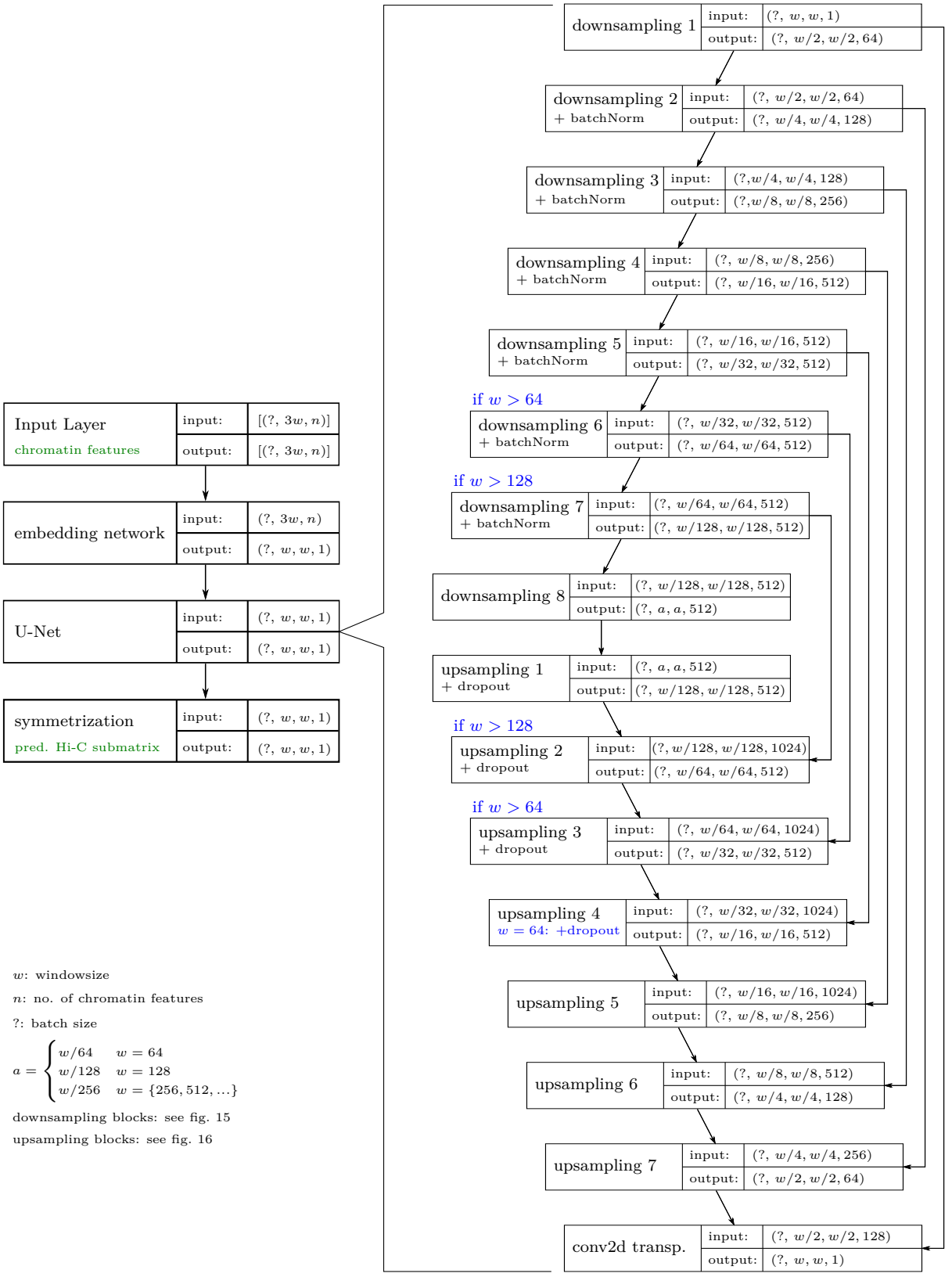


Figure 13: Adapted generator model from pix2pix

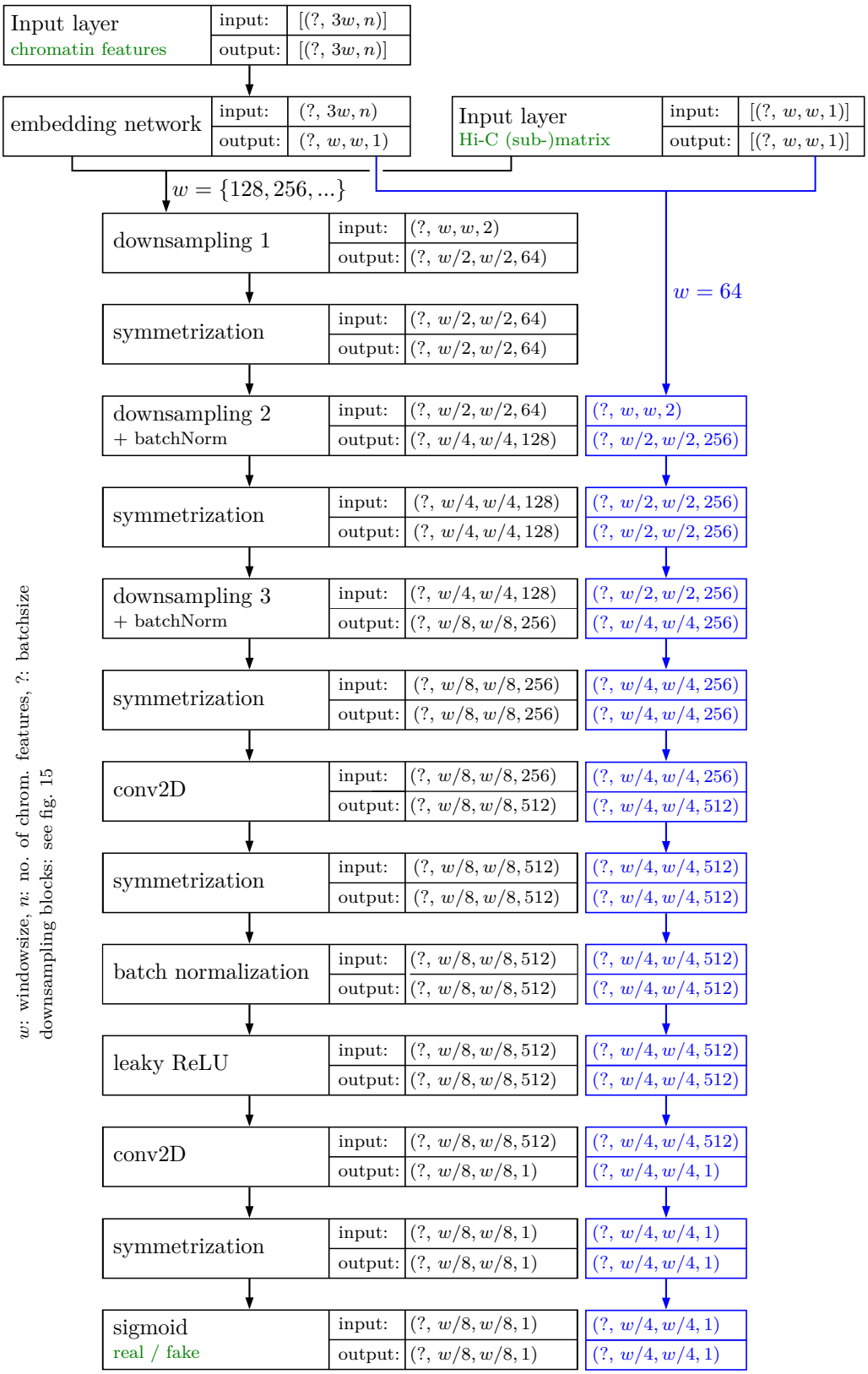


Figure 14: Adapted discriminator model from pix2pix

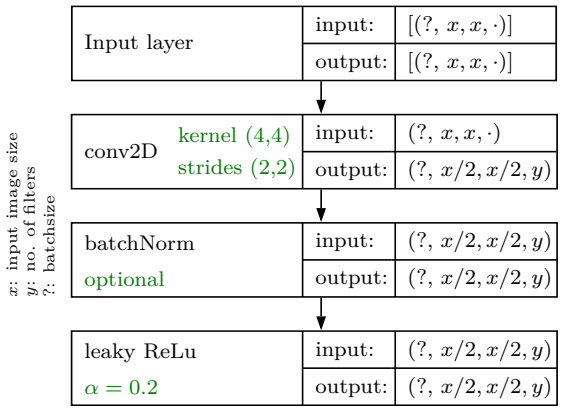


Figure 15: downsampling block

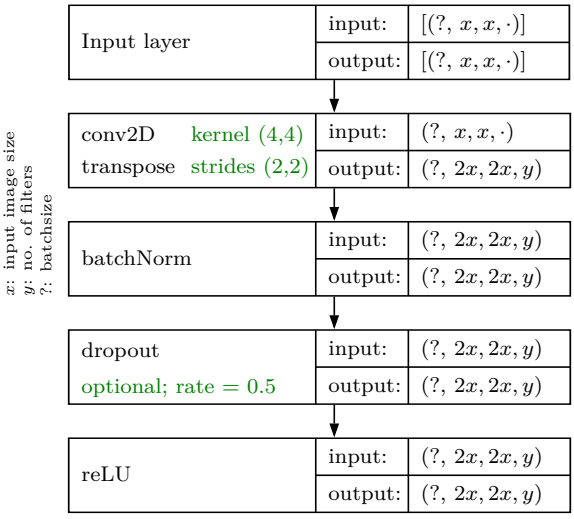


Figure 16: upsampling block

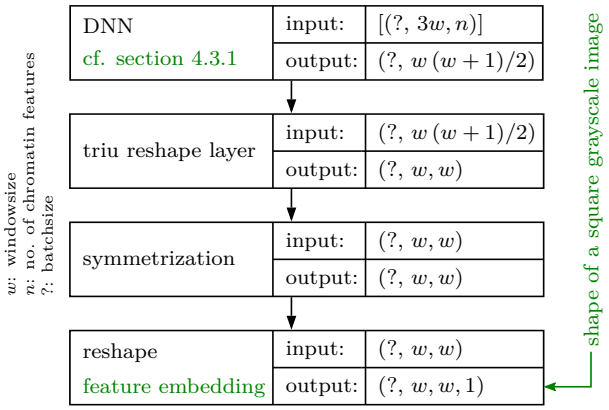


Figure 17: embedding network, DNN

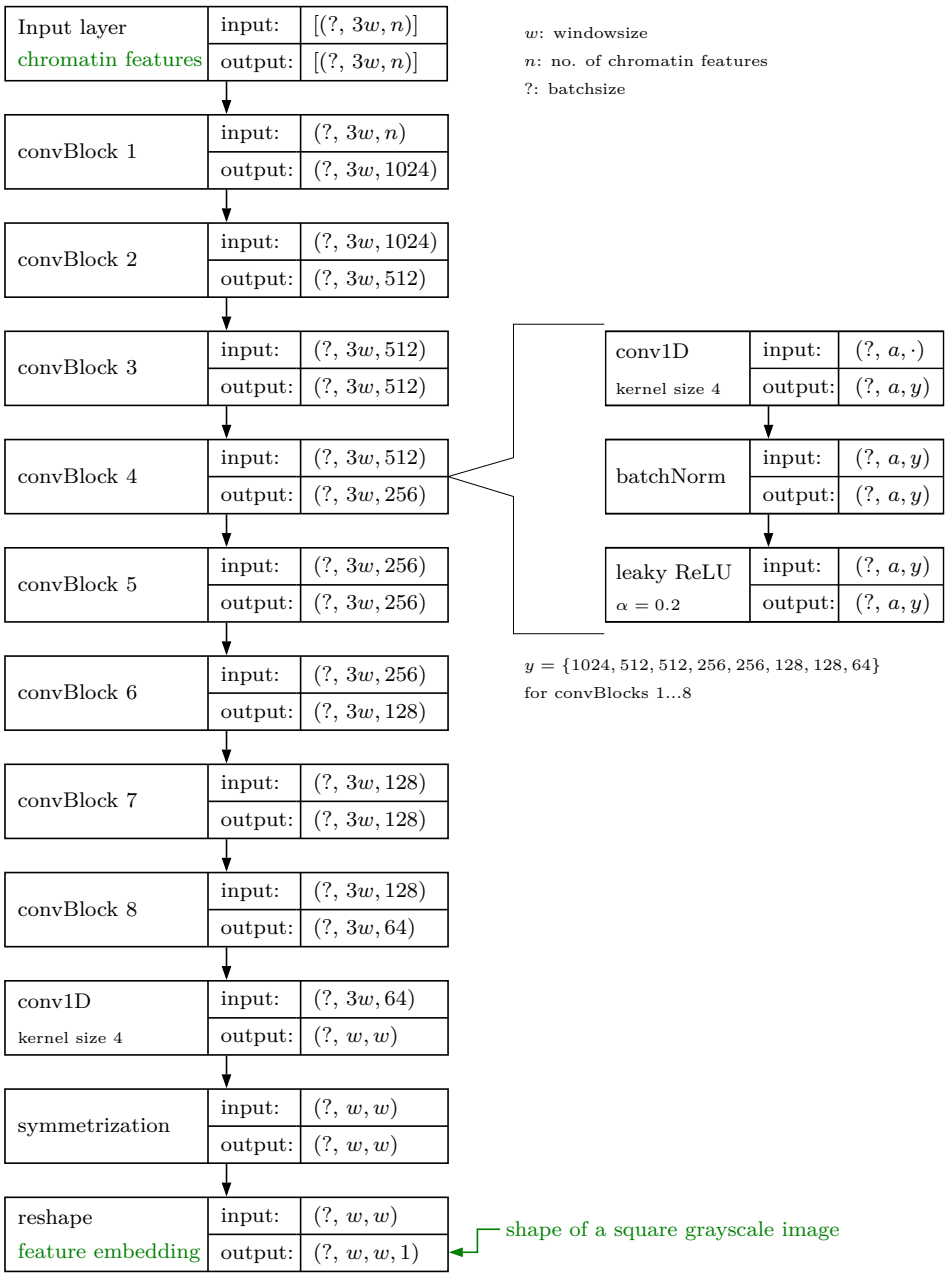


Figure 18: embedding network, CNN

5 Results

5.1 Dense Neural Network approaches

5.1.1 Initial results for comparison

The basic network was trained as explained in XXX. The validation error (MSE) for the basic neural network reached a minimum of about 150 000 after about 500 epochs for 25k binsize and about 400 epochs for 10k binsize, fig. 19f and 20f. Beyond that, the learning curve indicated overfitting, but this was often hard to spot in the matrix plots from the test set, compare e. g. fig. 22 and 23.

Figure 19 and fig. 20 show the Pearson correlations alongside area under the correlation curve (AUC) for the five test chromosomes at 25 and 10 kbp binsizes, respectively. The red lines in each correlation plot show the correlation between the corresponding training matrix from GM12878 and the target matrix from K562. It is obvious that all predicted test matrices have a strictly positive Pearson correlation, but are not better than transferring data from the training cell line.

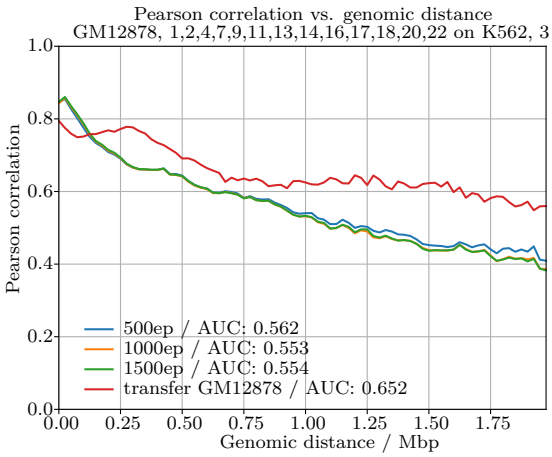
The predicted matrices themselves looked modest when plotted with pygenometracks XXX. While some of the highly interacting regions, for example between XXXand XXXof chromosome 19 were well predicted, other structures, especially larger ones like the one between XXXand XXXin chromosome 19 or between XXXand XXXin chromosome 21 were not predicted at all.

5.1.2 Results for variations of the convolutional part

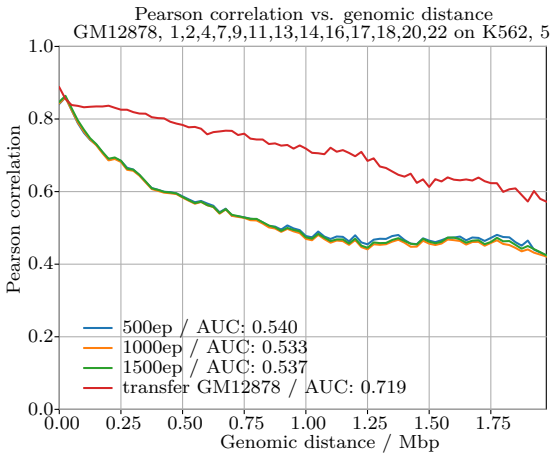
The predictions from the "wider" variant were generally similar to the initial results, both in terms of Pearson correlations and in terms of matrix plots, fig. 24 and XXX. Given the small increase in the number of trainable parameters at overall similar network topology, this is not surprising. However, it is interesting that hardly any improvement was found. Overfitting was less obvious than with the initial setup and the training process looked more smooth overall, but the remaining validation error was slightly higher than for the initial approach, fig. 24f.

The predictions from the "longer" variant were partially better for the test set than the initial ones in terms of Pearson correlations, fig. 25. Interestingly, no predictions were available for certain distances after 250 and 500 epochs, while predictions for all distances were available after 1000 epochs. The reason for this behavior is unknown, but due to the network setup, comparatively few neurons are responsible for certain distances. Since the longer network setup has considerably more trainable parameters, 500 epochs might not be enough to fully adjust the weights of these (outer) neuros. The learning process in itself looked more smooth and reached a lower validation error than before, fig. 25f.

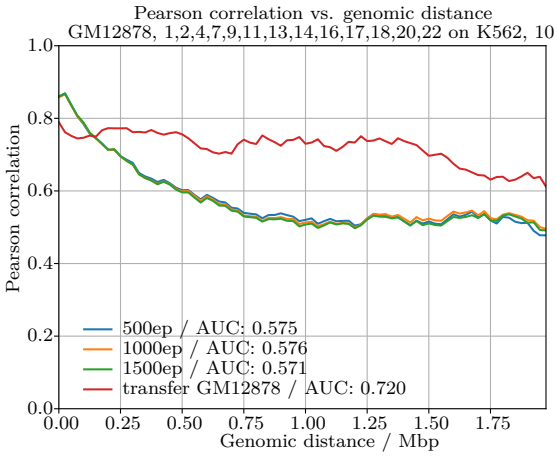
The Pearson correlations for predictions from the "wider-longer" variant are shown in fig. 26. While improvements can be seen for 3 of 5 test chromosomes compared to the initial network, the results were worse than the ones from the highly similar "longer"-variant alone, and the remaining validation error was also higher.



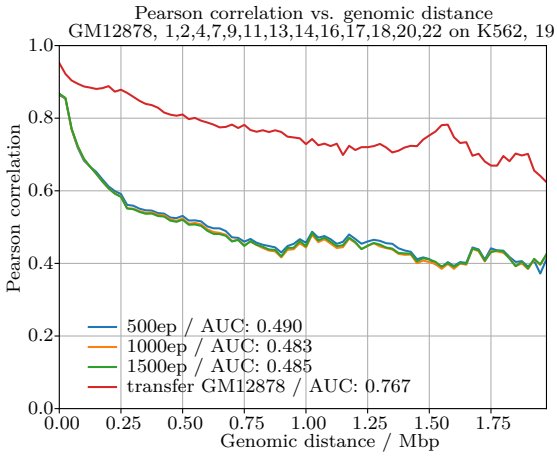
(a) chr3



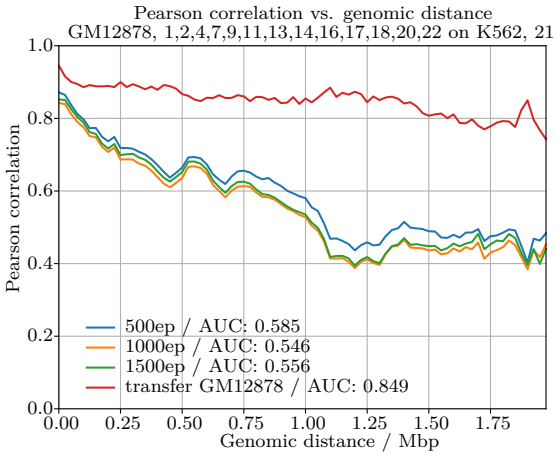
(b) chr5



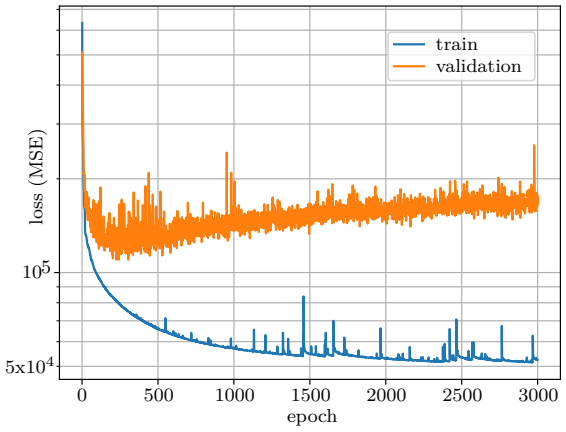
(c) chr10



(d) chr19

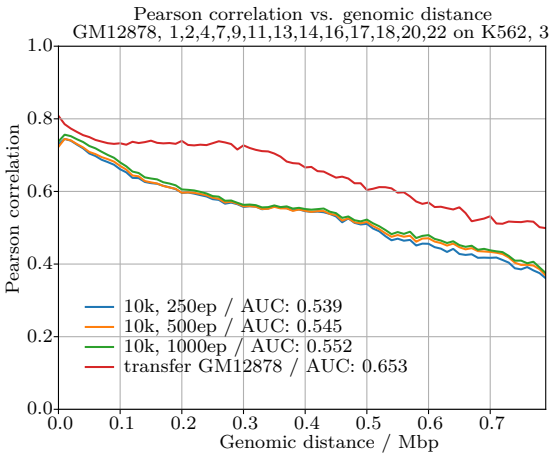


(e) chr21

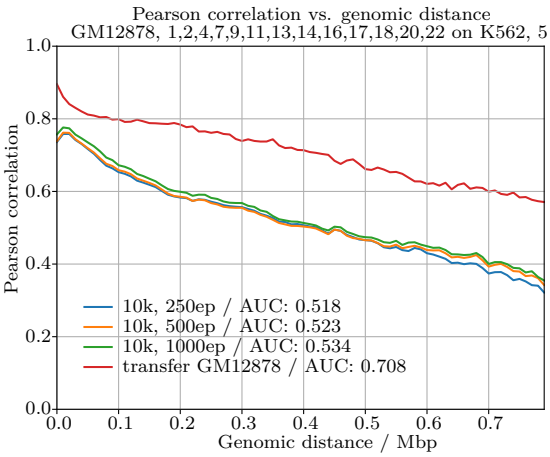


(f) learning progress

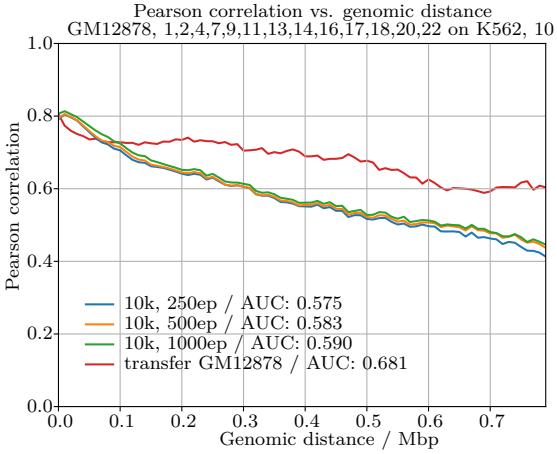
Figure 19: results / metrics, basic DNN, 25 kbp, test chromosomes



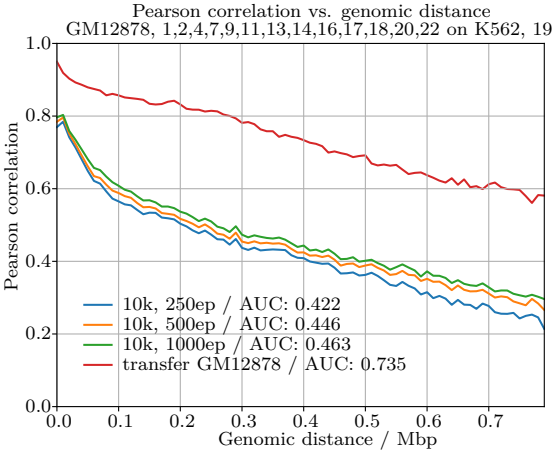
(a) chr3



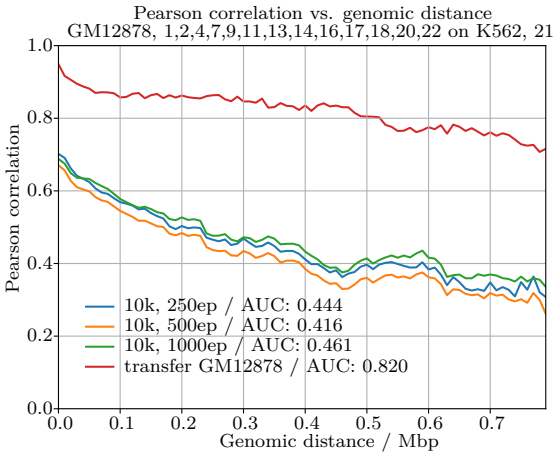
(b) chr5



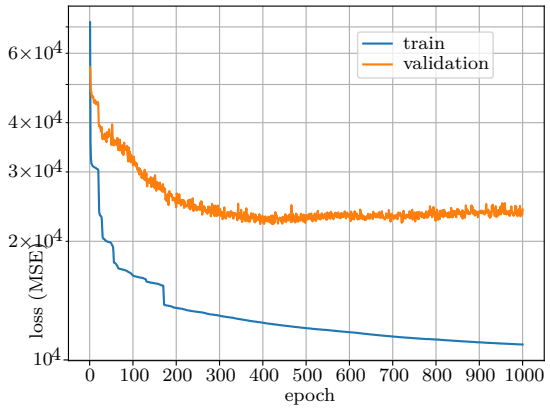
(c) chr10



(d) chr19



(e) chr21



(f) learning progress

Figure 20: results / metrics, basic DNN, 10 kbp, test chromosomes

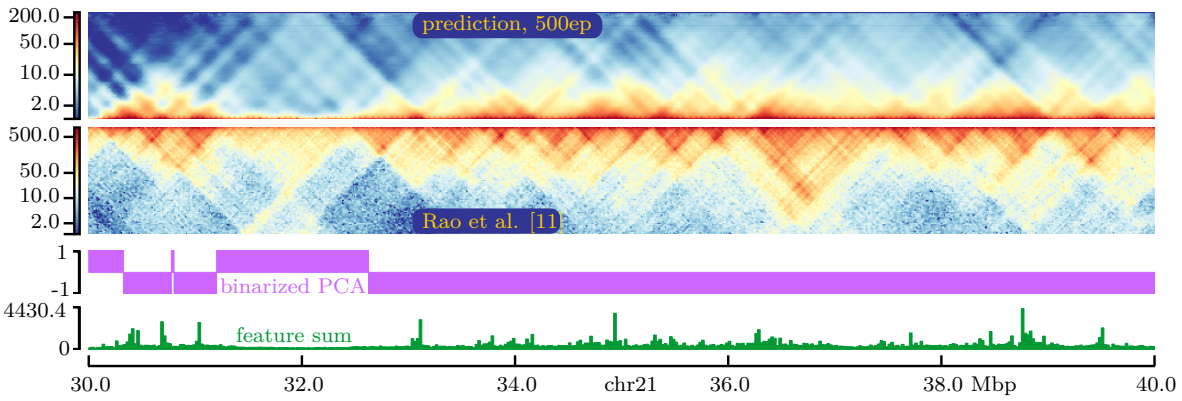


Figure 21: example prediction, 500 epochs

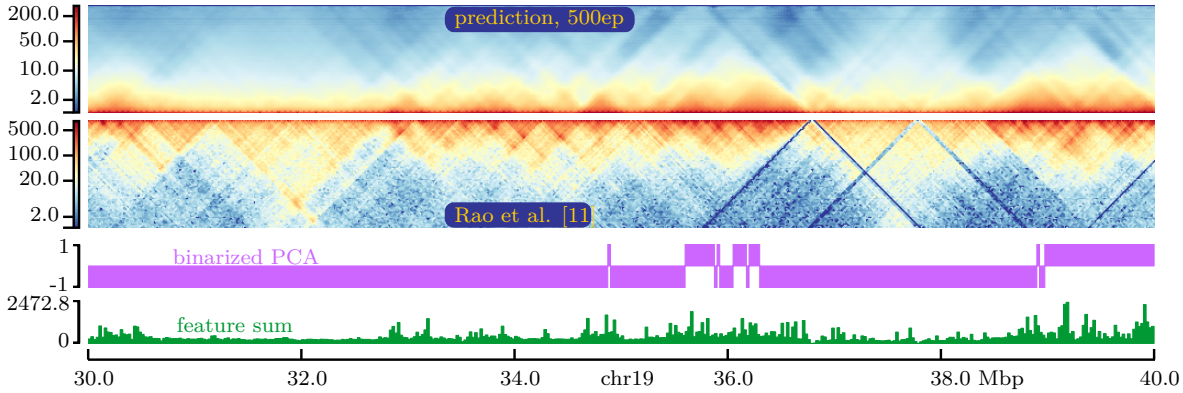


Figure 22: example prediction, 500 epochs

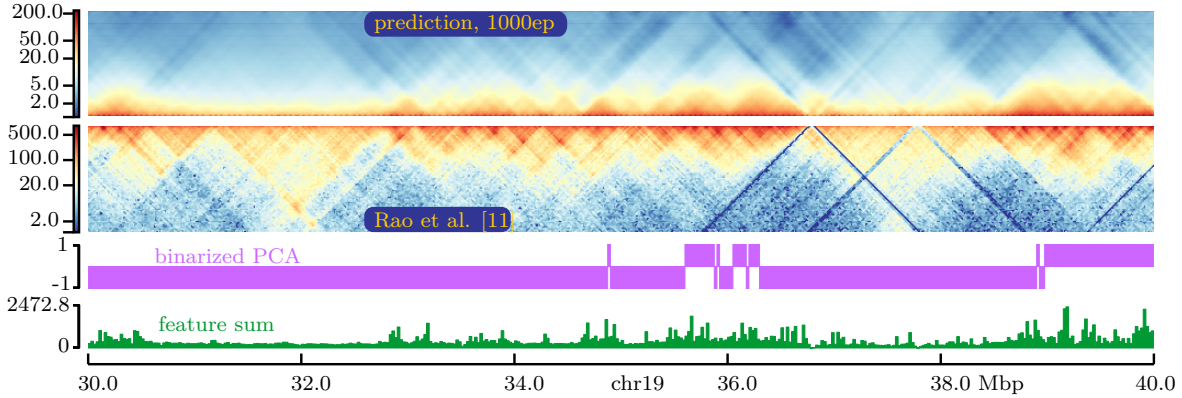
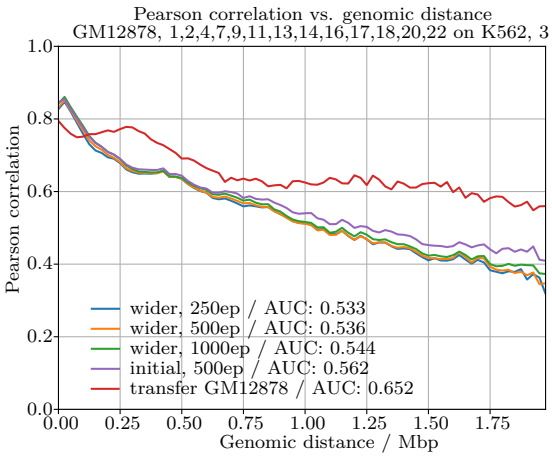
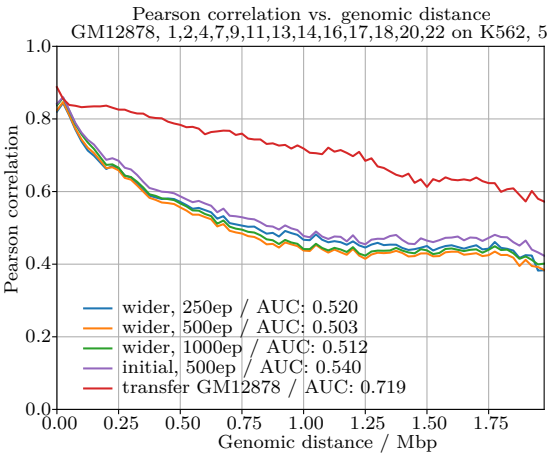


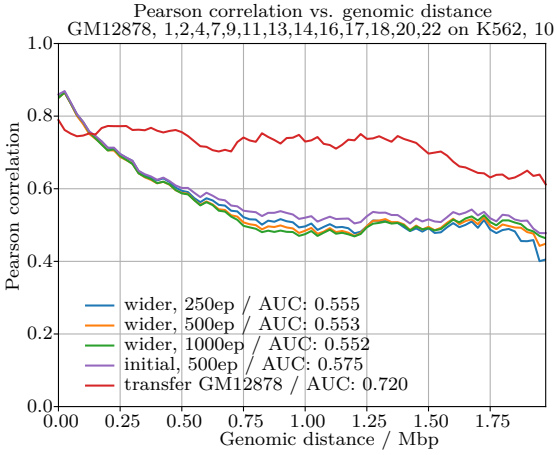
Figure 23: example prediction, 1000 epochs



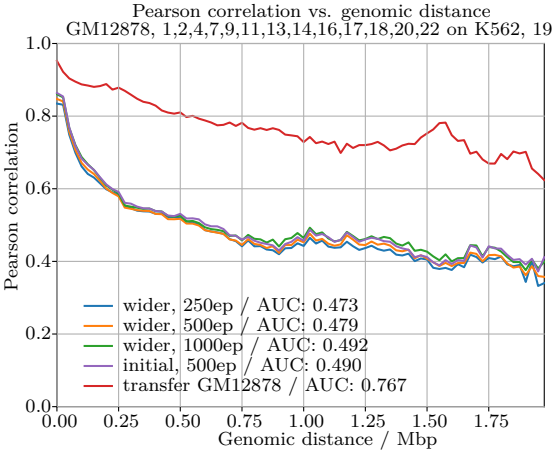
(a) chr3



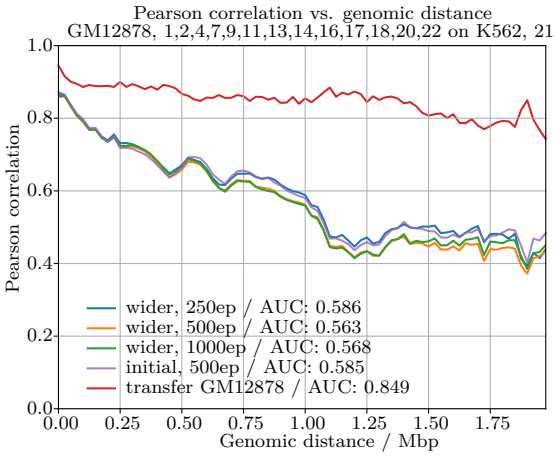
(b) chr5



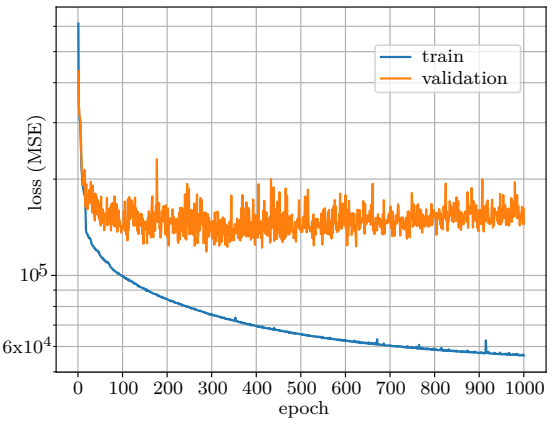
(c) chr10



(d) chr19

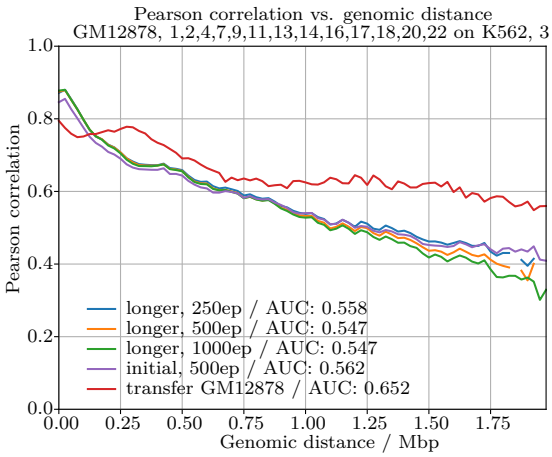


(e) chr21

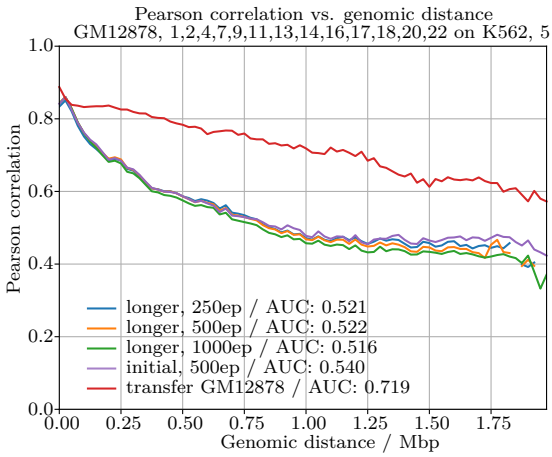


(f) learning progress for wider DNN

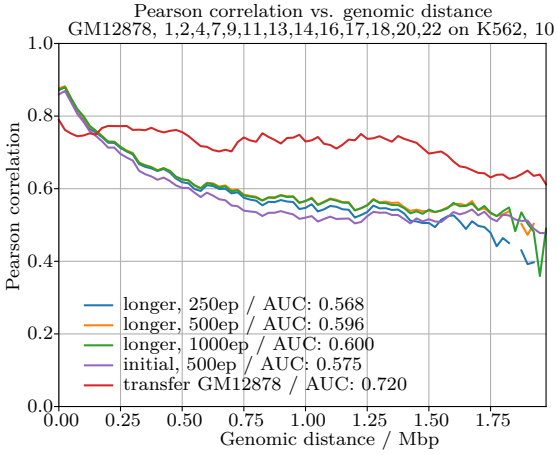
Figure 24: Pearson correlations, “wider” variant of DNN, test chromosomes



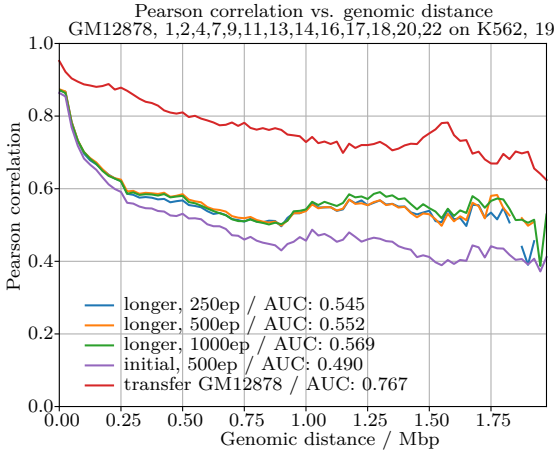
(a) chr3



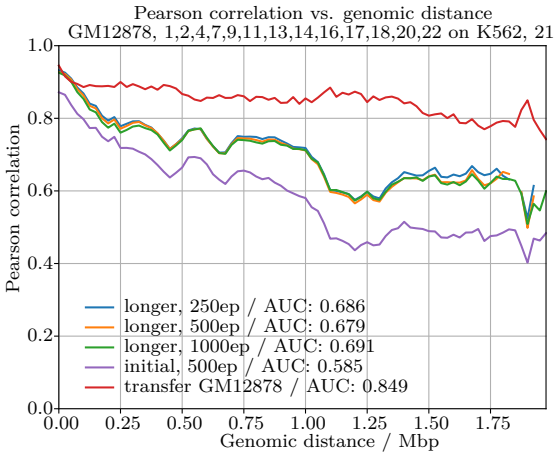
(b) chr5



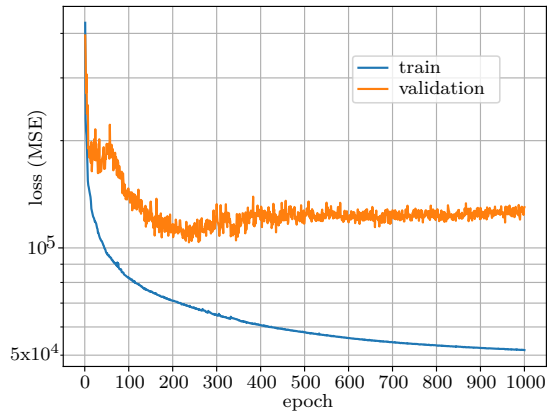
(c) chr10



(d) chr19

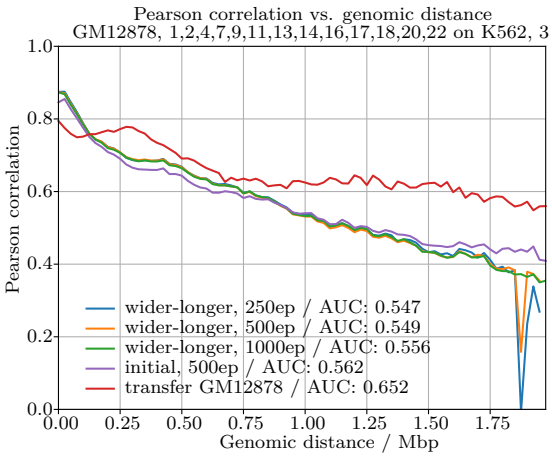


(e) chr21

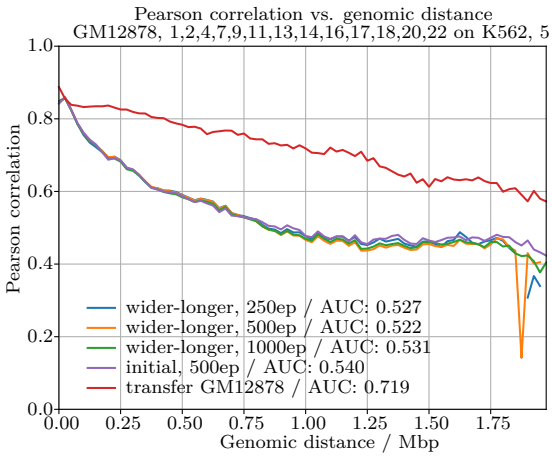


(f) learning progress for longer DNN

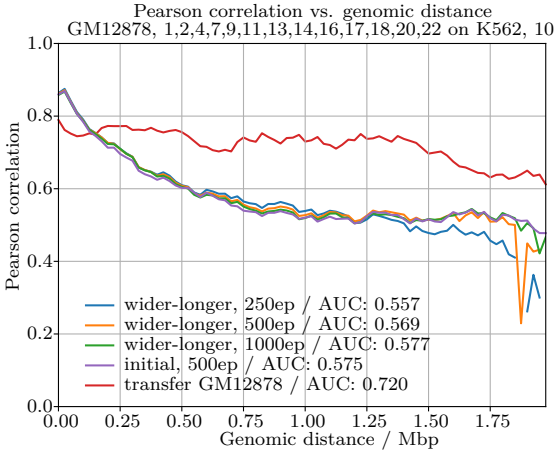
Figure 25: results / metrics, “longer” variant of DNN, test chromosomes



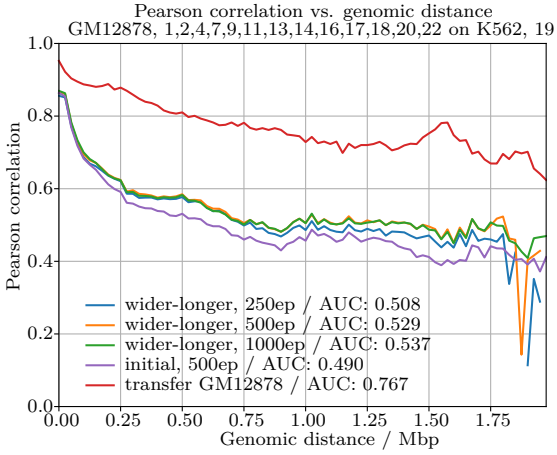
(a) chr3



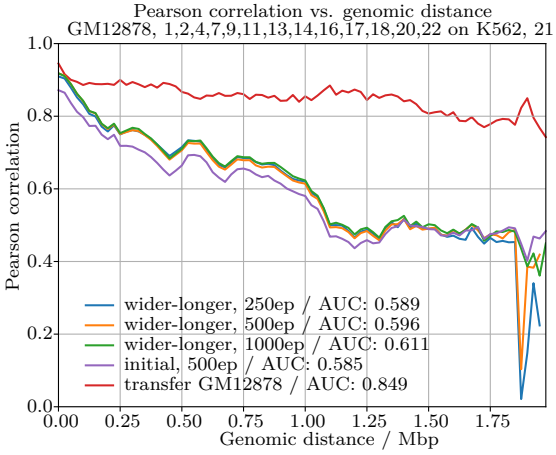
(b) chr5



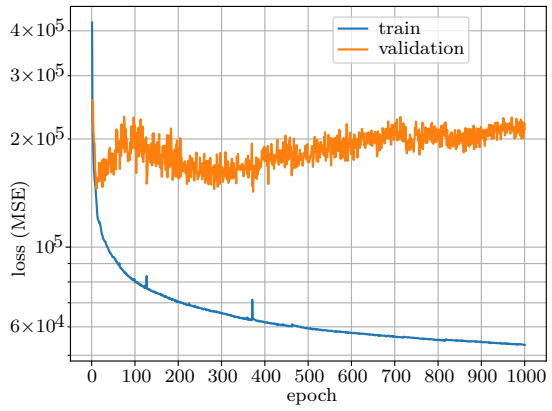
(c) chr10



(d) chr19



(e) chr21



(f) learning progress

Figure 26: results / metrics, "wider-longer" variant of DNN, test chromosomes

The Pearson correlations for the variant with feature binsize 5 kbp and matrix binsize 25 kbp are shown in fig. 27. Much like the "wider" variant, the results did not improve compared to the initial predictions. The learning curve was smooth and showed signs of slight overfitting beyond 300 epochs, fig. 27f.

5.1.3 Results for combined loss function

The Pearson correlations for a combined loss function according to eq. (9) with weighting parameters $\lambda_{MSE} = 0.8999, \lambda_{VGG} = 0.1, \lambda_{TV} = 0.0001$ are shown in fig. 28. For all test chromosomes, the results were highly similar to the inital network's results.

While manually finding parameters λ for MSE- and VGG-loss that made the results better was not successful, it was found that the TV loss weight needed to be much smaller than the two other weights. Otherwise, many true interactions outside the first few matrix diagonals were considered as noise and optimized away early in the training process. XXXmaybe put one figure here from `2020-12-05_tvLoss`

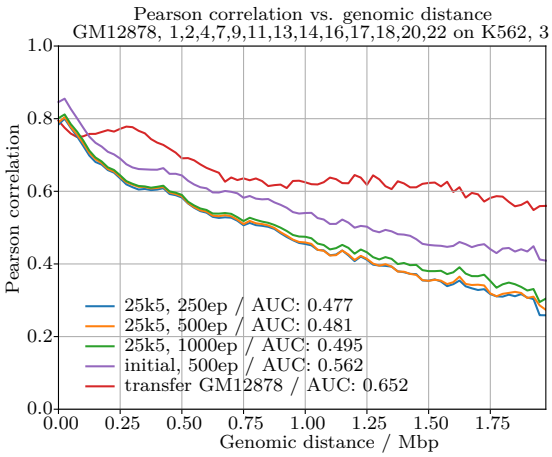
5.1.4 Results for score-based loss function

The Pearson correlations for the DNN with score-based loss function with parameters $\lambda_{MSE} = 1.0$, $\lambda_{score} = 100$, $ds = 12$ are shown in figure 29. While a slight improvement was achieved for test chromosome 21, the Pearson correlations of the others remained widely unchanged, but at about 7 min per epoch, the training time was much longer than for the initial network. It is well possible that a stronger improvement *can* be achieved by targeted parameter tuning. However, the results achieved by manual parameter tuning were also not encouraging towards a tedious grid- or tree-search.

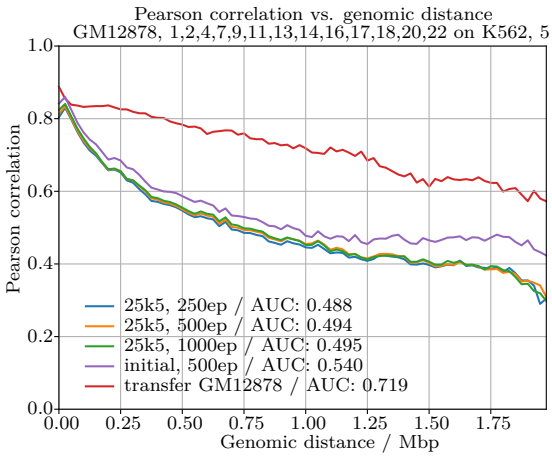
5.1.5 Results for different binsizes and windowsizes

Training the network with matrix- and feature binsizes of 50 kbp ("50k direct") did not improve the results in the desired way, fig. 30. Larger structures in the test matrices were not more prominent than before, XXX, and simply coarsening the results from 25 kbp yielded better results in all test regions except chromosome 21. The same held for re-using the initial network trained at 25 kbp to predict at 50 kbp, fig. 30 ("initial 25k→50k"). Additionally, the training process for 50k collapsed after about 420 epochs for unknown reasons – which was not considered too problematic here, because the optimum validation error had already been reached between 150 and 250 epochs, fig. 30f. Faster convergence in itself would not be surprising, since there are only about half as many training samples at 50 kbp compared to 25 kbp, cf. table 3.

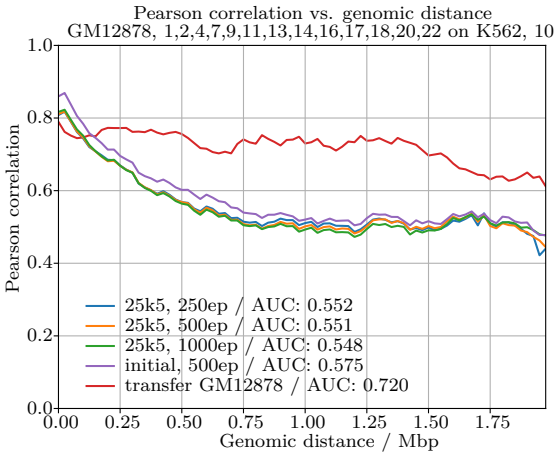
Simultaneously training a network with matrix- and feature binsizes of 25 kbp and 50 kbp turned out unproblematic with regard to convergence, fig. 31f, but the predictions at both 25 kbp and 50 kbp were – often significantly – worse than the initial predictions at that binsize, fig. 30 ("25k+50k→50k") and fig. 31 ("25k+50k→25k").



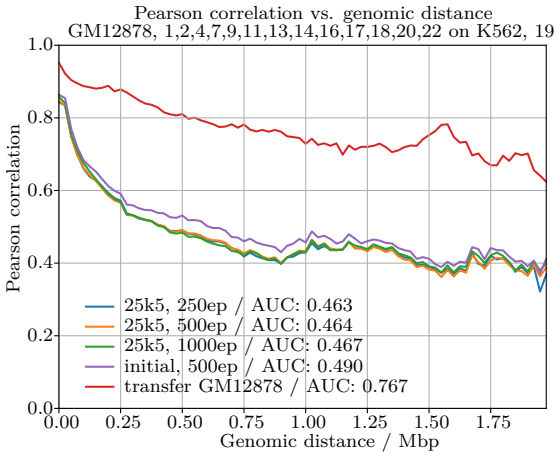
(a) chr3



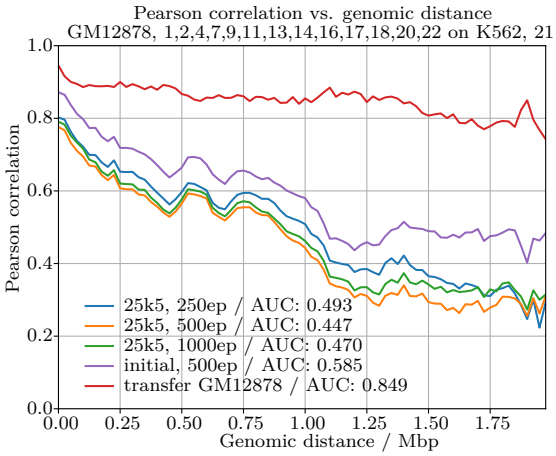
(b) chr5



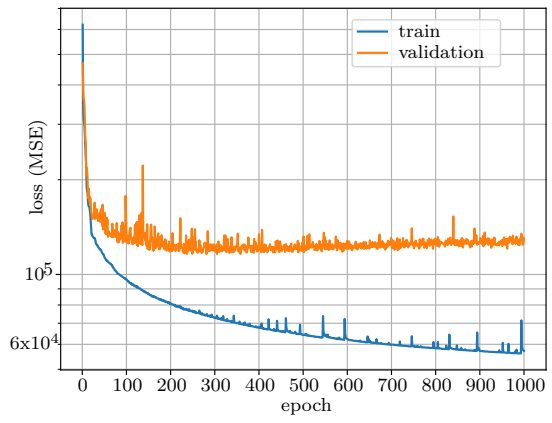
(c) chr10



(d) chr19

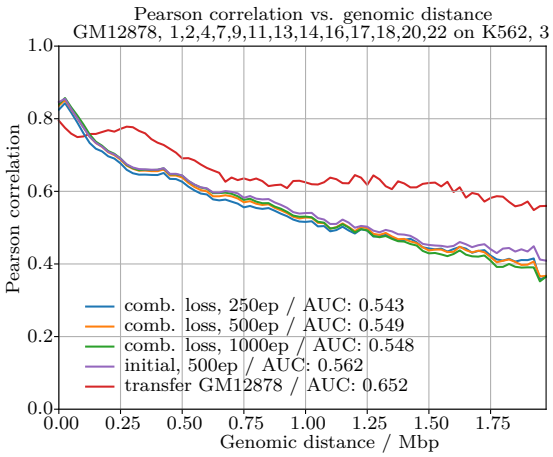


(e) chr21

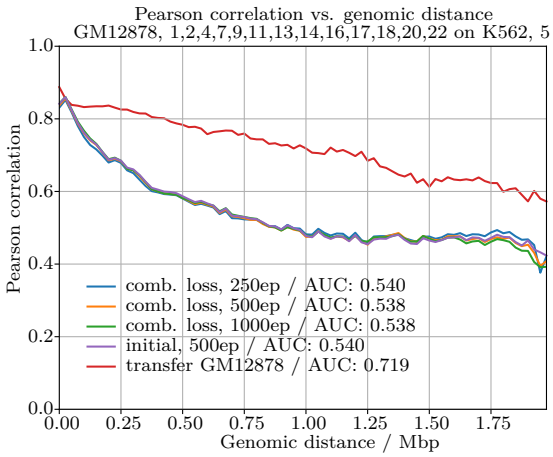


(f) learning progress

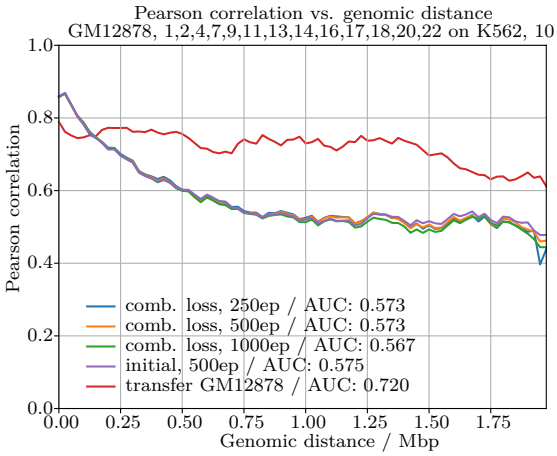
Figure 27: results / metrics, "5k – 25k" variant of DNN with $b_{feat} = 5\,\text{kbp}$ and $b_{mat} = 25\,\text{kbp}$, test chromosomes



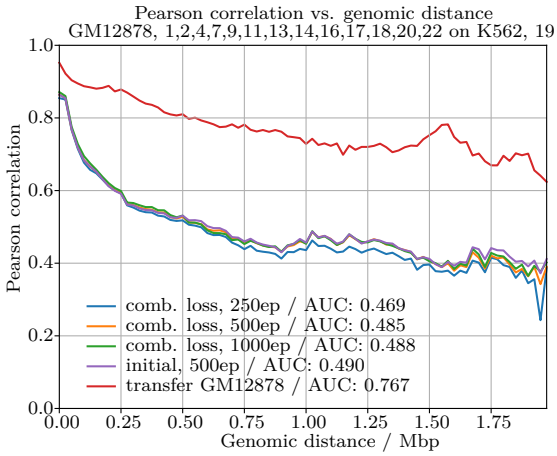
(a) chr3



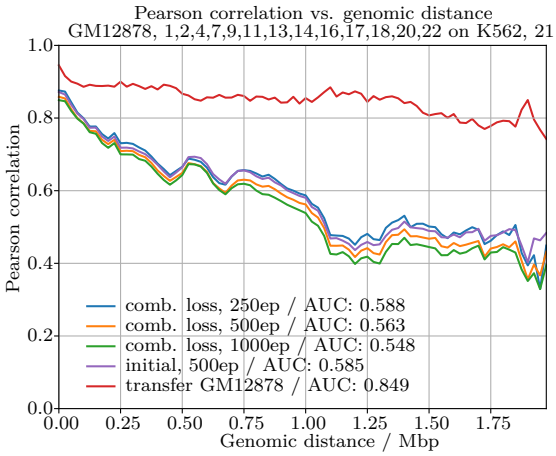
(b) chr5



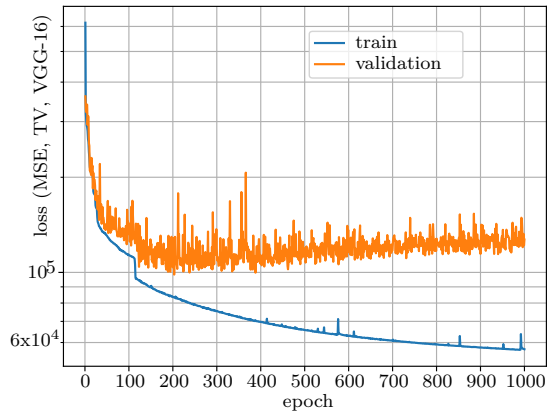
(c) chr10



(d) chr19

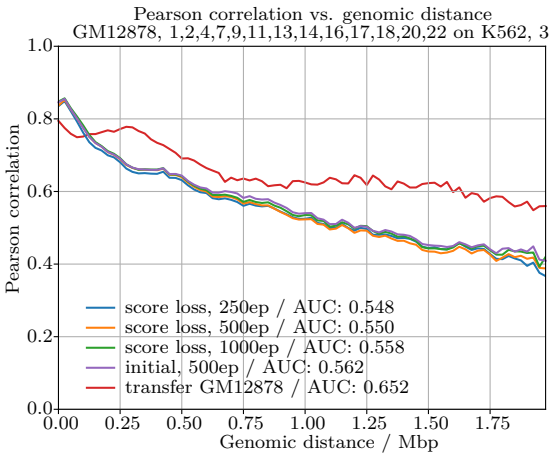


(e) chr21

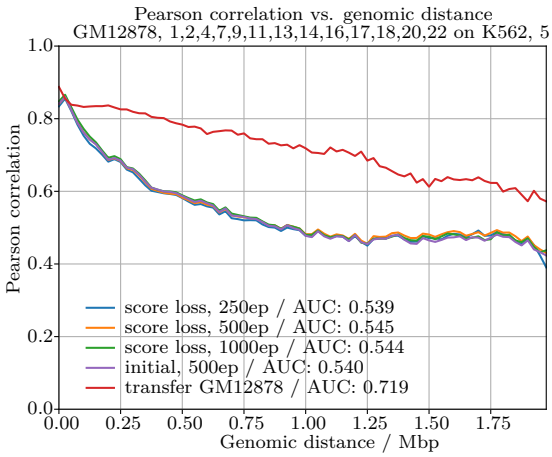


(f) learning progress

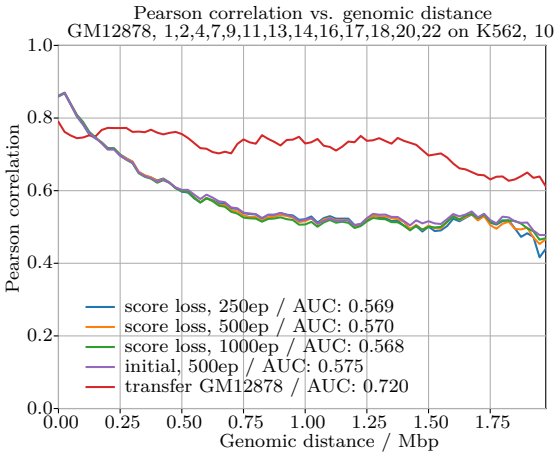
Figure 28: results / metrics, DNN with combined loss function (MSE, TV, VGG-16), test chromosomes



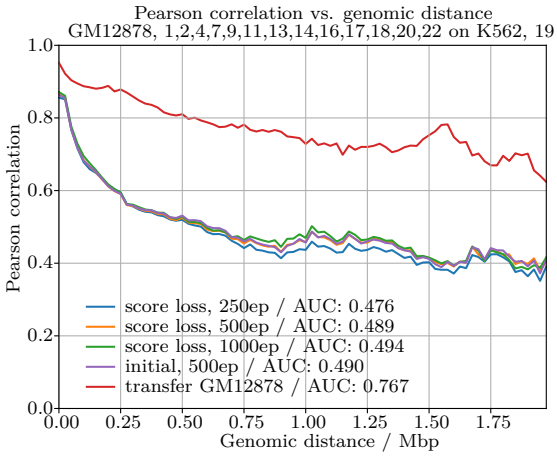
(a) chr3



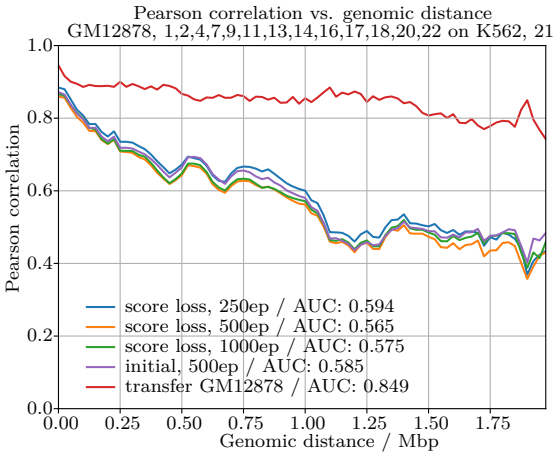
(b) chr5



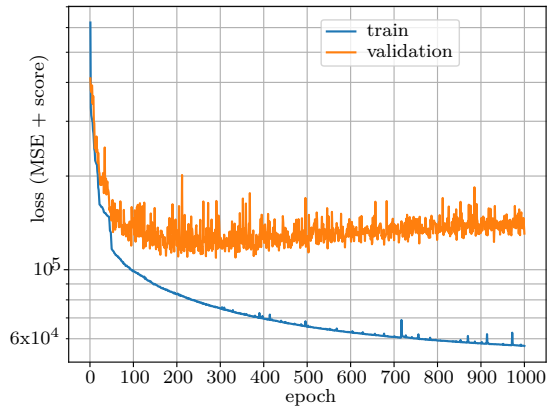
(c) chr10



(d) chr19

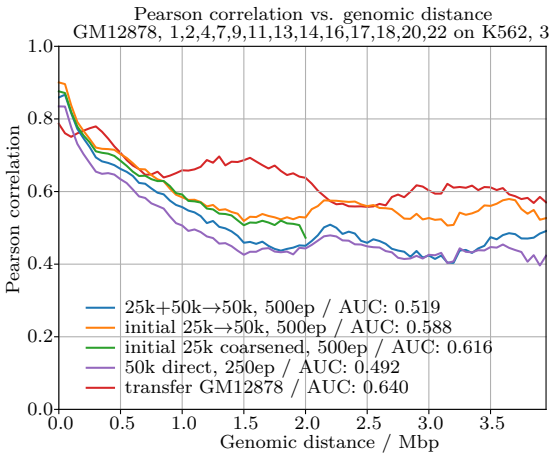


(e) chr21

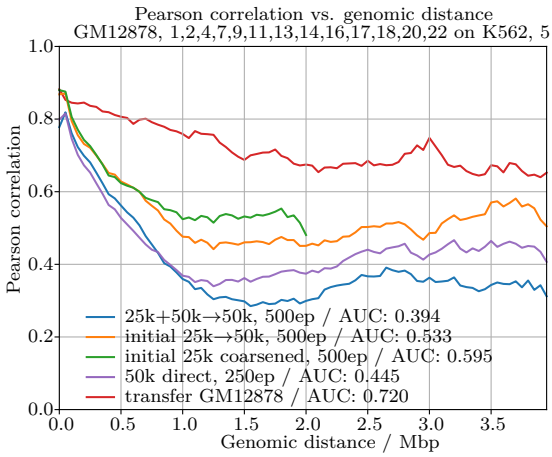


(f) learning progress

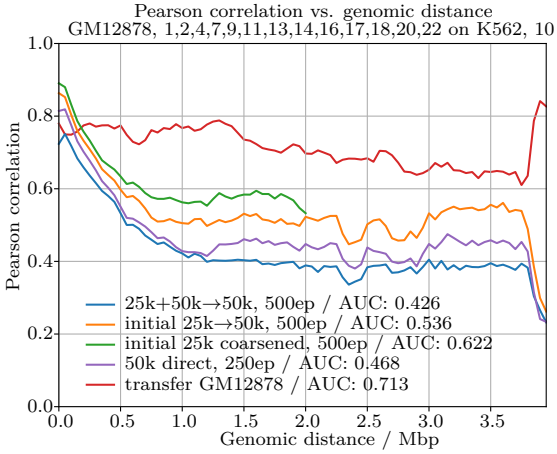
Figure 29: results / metrics, DNN with score-based loss function, test chromosomes ($\lambda_{MSE} = 1.0,\ \lambda_{score} = 100,\ ds = 12$)



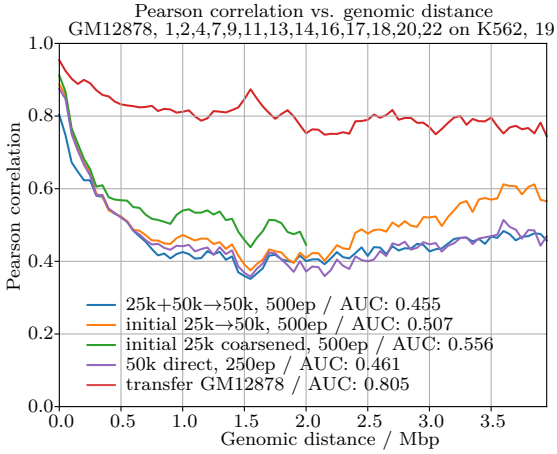
(a) chr3



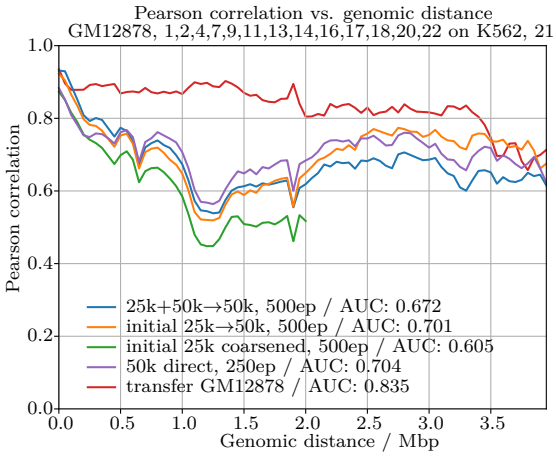
(b) chr5



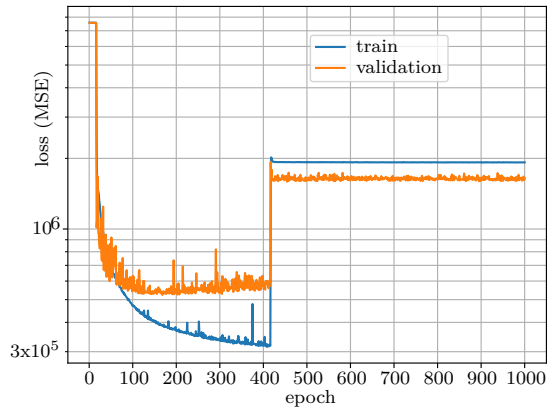
(c) chr10



(d) chr19

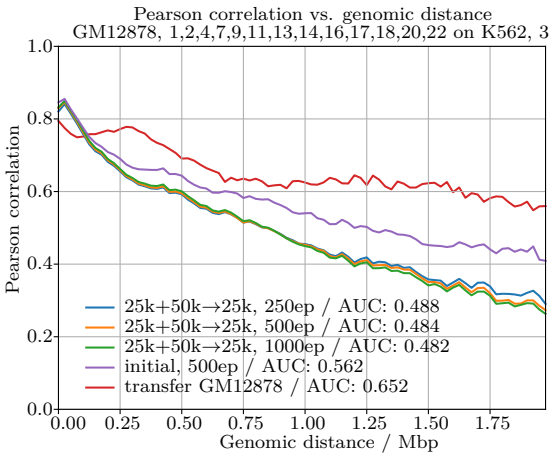


(e) chr21

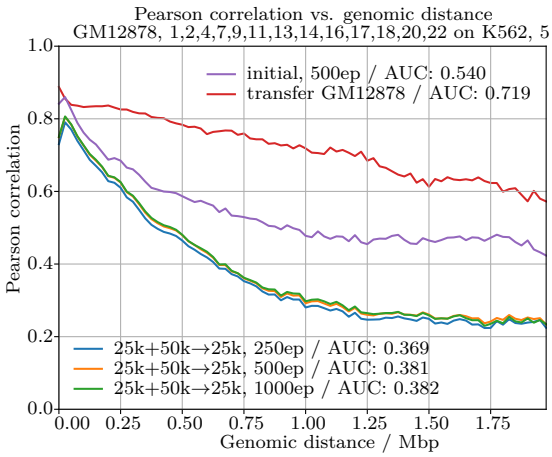


(f) learning progress 50k direct

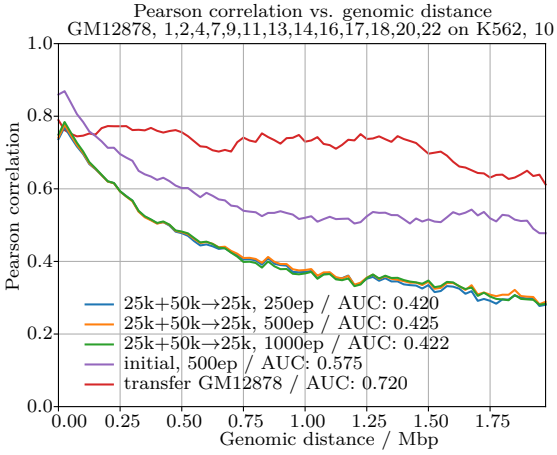
Figure 30: results / metrics, various DNNs at 50 kbp



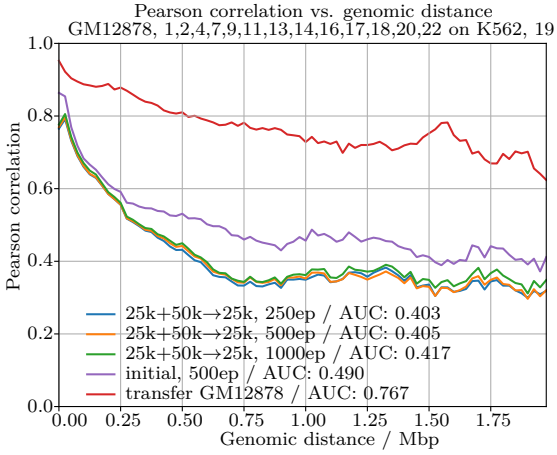
(a) chr3



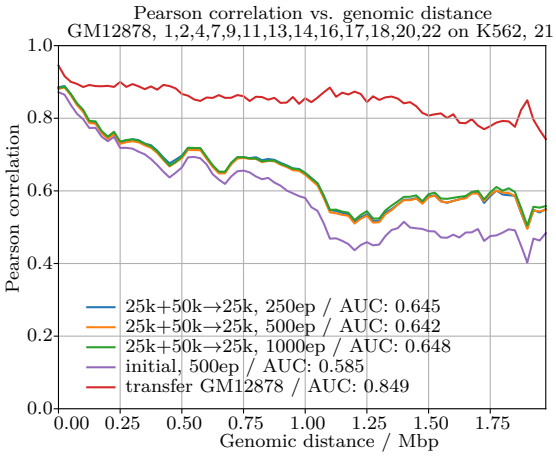
(b) chr5



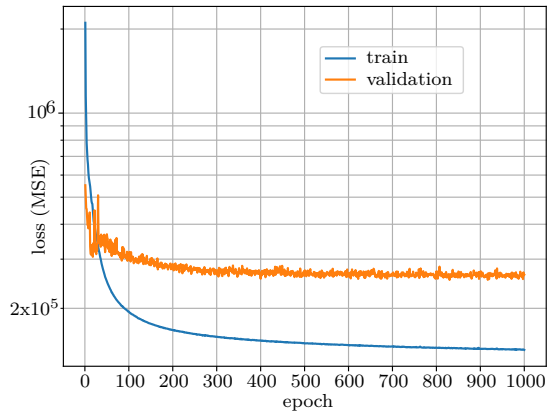
(c) chr10



(d) chr19



(e) chr21



(f) learning progress

Figure 31: results / metrics, DNN trained at 25 kbp and 50 kbp simultaneously

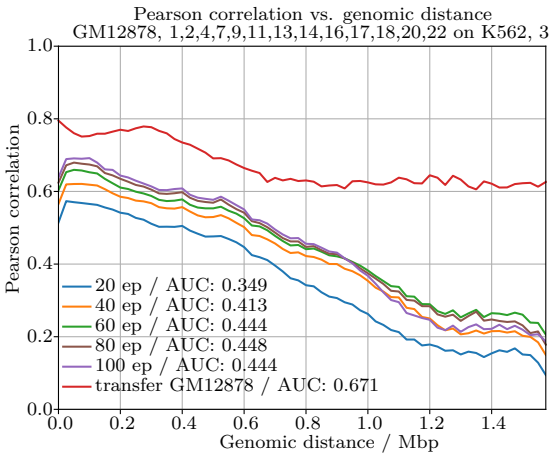
5.2 Hi-cGAN approaches

5.2.1 cGAN with DNN embedding

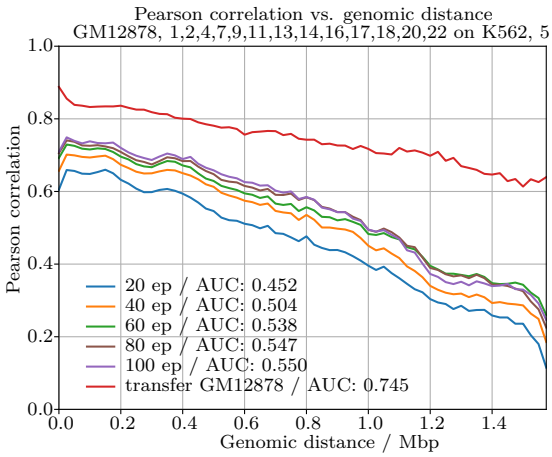
5.2.2 cGAN with CNN embedding

The results from the cGAN were generally better than the best results from the DNN, and from windowsize 128, they were also close to the baseline or better. Interestingly, acceptable results were obtained already after only 25 epochs. Fast convergence is well known from pix2pix [48], but it is still surprising that this property was maintained despite the changes made to the original network.

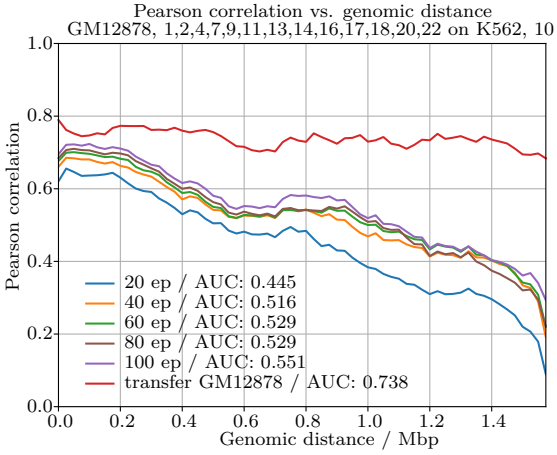
For windowsizes 64 and 128 bins, the optimal number of epochs seemed to be around 80, while for windowsize 256, a number greater 100 epochs might have been favorable. However, this would have come at a large computation time, since average training time was around 108 min per epoch on the given hardware.



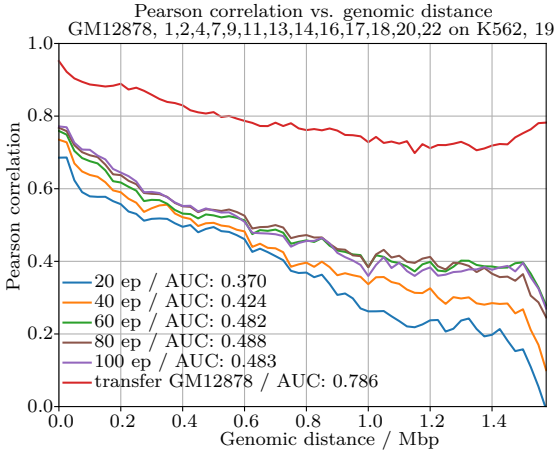
(a) chr3



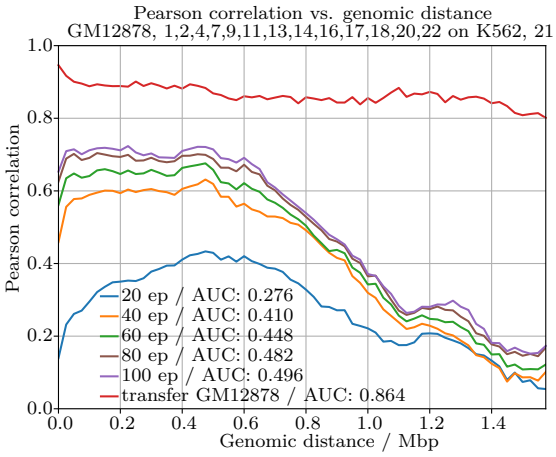
(b) chr5



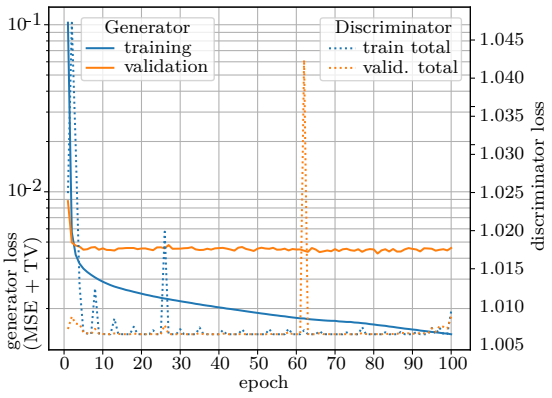
(c) chr10



(d) chr19

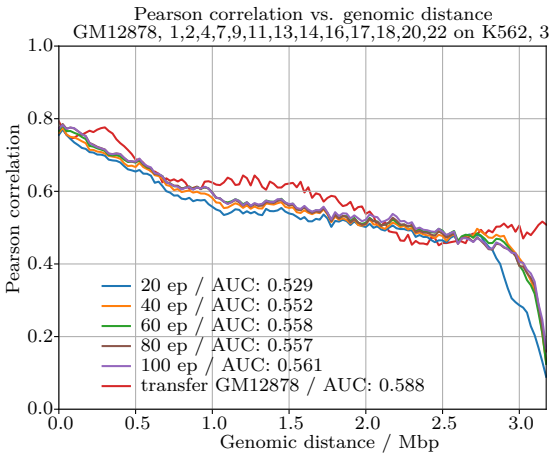


(e) chr21

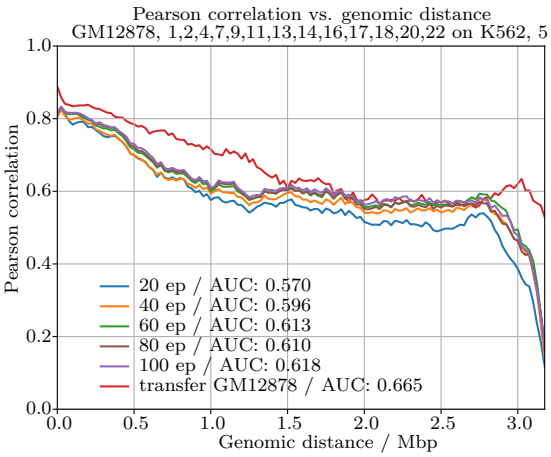


(f) learning progress

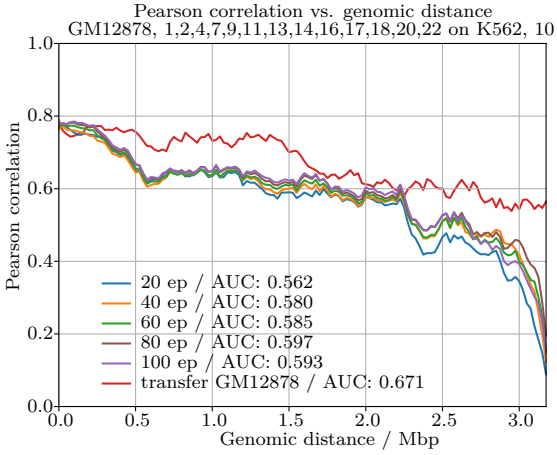
Figure 32: results / metrics cGAN, windowsize 64, test chromosomes



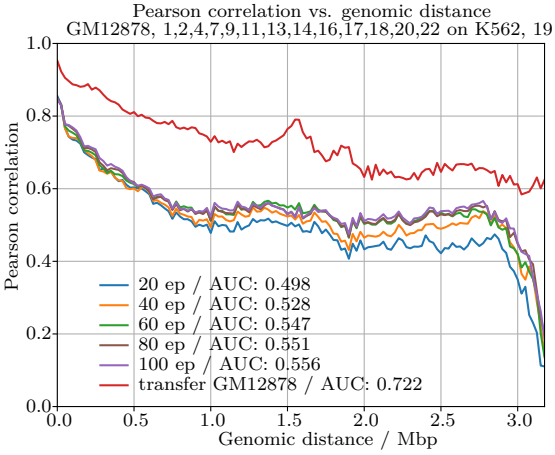
(a) chr3



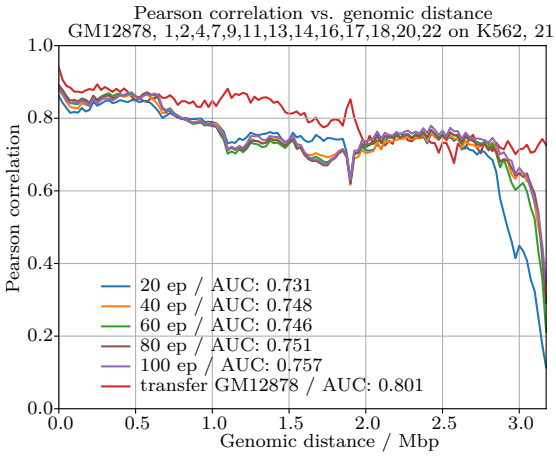
(b) chr5



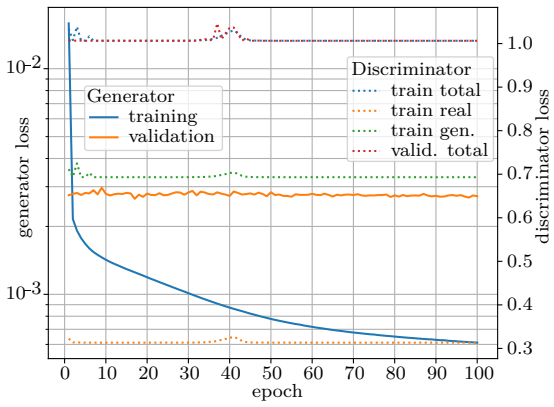
(c) chr10



(d) chr19

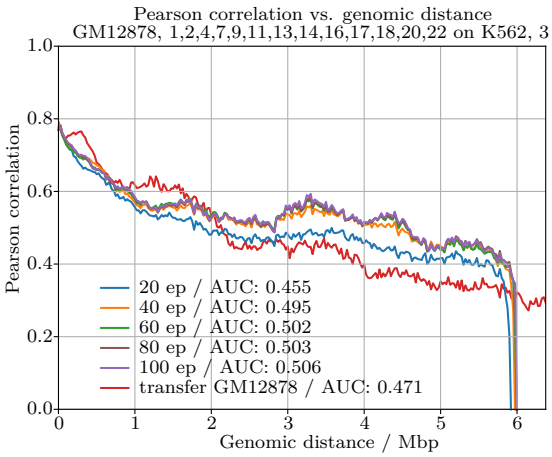


(e) chr21

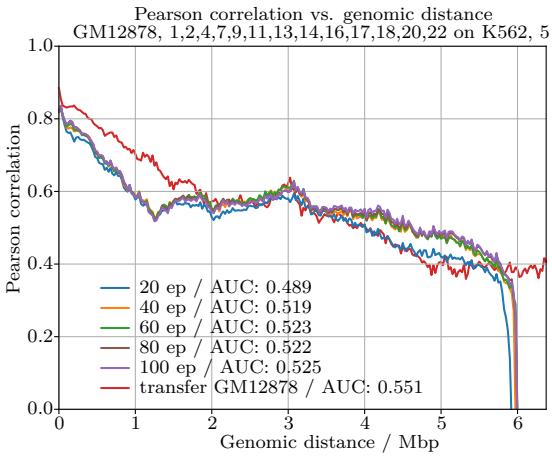


(f) learning progress

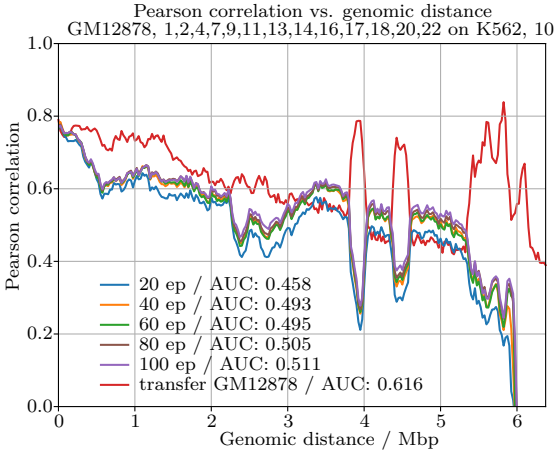
Figure 33: results / metrics cGAN, windowsize 128, test chromosomes



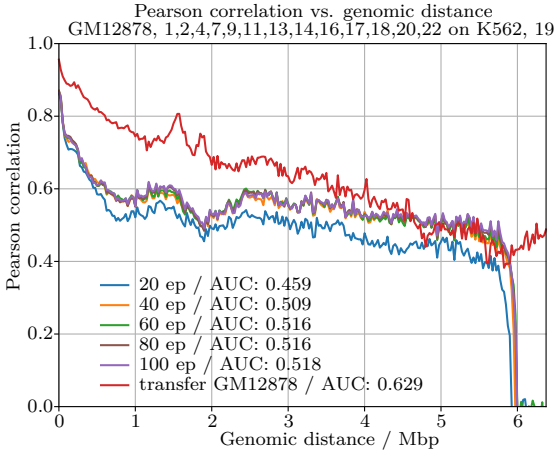
(a) chr3



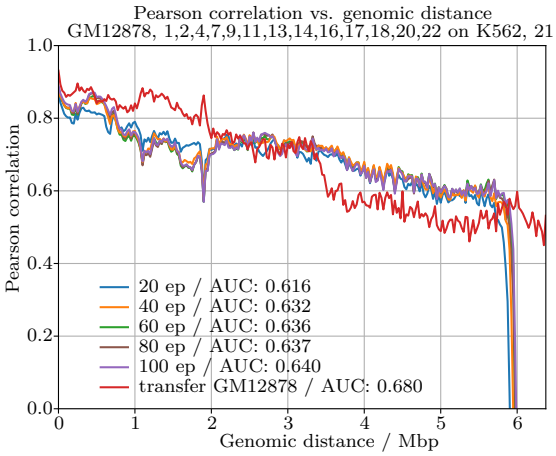
(b) chr5



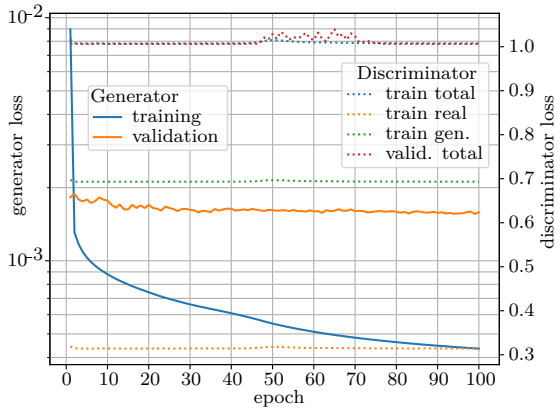
(c) chr10



(d) chr19



(e) chr21



(f) learning progress

Figure 34: results / metrics cGAN, windowsize 256, test chromosomes

6 Discussion and Outlook

Approach by Farre et al also working for human data. Changes might bring benefits in certain cituations, but are not a big improvement.

cGAN seems promising, but the discriminator remains to improve. Improvements still not good enough to beat transfer GM12878-K562 in many cases.

If possible, compare to other approaches. Maybe we can use the one by Zhang et al. for which data from study project exists, but it might take too long.

Learning from mixed up samples, true matrix but non-matching conditional input could improve things, commonly done in text-to-image synthesis networks.

7 Appendix

7.1 Chromatin feature download links

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

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

7.3 Hardware

	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 5: key figures of hardware used throughout the thesis

Note: CPU frequency in GHz; L1,L2,L3 cache in KB; GPU memory in MiB; RAM (main memory) in GB

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] 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.

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

[50] 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.

[51] 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.

[52] 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.

[53] Justin Johnson, Alexandre Alahi and Li Fei-Fei. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”. In: (Mar. 27, 2016). arXiv: 1603.08155 [cs.CV].

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

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

[65] 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.

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

[67] 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.

[68] 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.

[69] 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.

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

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

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

[73] 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).

Acronyms

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.