Universiteit Leiden

Opleiding Informatica

Convolutional Neural Networks for Regulatory Genomics

Name:	Niels ten Dijke
Date:	17/06/2017
1st supervisor:	dr. Wojtek Kowalczyk
2nd supervisor:	dr. Gerard van Westen
ext. supervisor:	dr. Daniel Zerbino

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

CONVOLUTIONAL NEURAL NETWORKS FOR REGULATORY GENOMICS

Niels ten Dijke

A thesis submitted for the degree of Master of Science

17/06/2017

Abstract

A majority of the human genome consists of sequences that do not code for a particular protein, so called non-coding DNA. The non-coding regions nonetheless play a vital role in gene expression. These non-coding regions of the DNA contain cis-regulatory elements such as promoters and enhancers. These regions can be bound by transcription factor proteins and thereby controlling the rate of transcription of DNA to messenger RNA. This then helps to regulate the expression of nearby genes.

Next-generation sequencing (NGS) techniques allow for identifying and studying the genomic factors such as transcription factor binding, histone modifications and open chromatin that underlie transcription with great sequencing depth. Furthermore, these data allow researchers to build predictive models for these events using machine learning approaches, which permit the annotation of new cell types without having to perform the experiment. In particular, convolutional neural networks seem to be well suited to model genomic data. A convolutional neural network (CNN) is a type of feed-forward neural network inspired by the animal visual cortex. CNNs are characterized by having spatially local connections. This connectivity pattern allows CNNs to be effective on data that have a grid-like topologies. In other words, data that can be represented by nodes which are connected to neighbors along one or more dimensions, where neighboring elements have statistical dependencies. Recently, algorithmic advances as well as great improvements in processing capabilities and tools and better datasets have made it possible to train increasingly complex models. Indeed, deep convolutional neural networks have proven to be very successful on many artificial intelligence tasks such as image classification, finding policy and value functions for game playing AI and drug discovery. As for typical NGS data, which includes DNA sequences, open chromatin and transcription factor binding data, these are all one dimensional grids.

Identifying transcription factor binding sites can greatly help researchers understand the transcription process and the underlying factors to genetic diseases. In the first experiment, convolutional neural networks models were built to predict transcription factor binding sites from sequence, open chromatin, gene expression and DNA shape data. We found the convolutional neural network to perform close to the state of the art on some transcription factors, while performing significantly worse on others. Building models for each task separately resulted in better predictive performance than a multi-task network modeling all transcription factors simultaneously.

In the second experiment, we took a closer look at the transcription process. The exact location of transcription initiation, the transcription start site (TSS), can be determined experimentally at base pair resolution. Unlike translation, where the exact amino acid triplet for starting the translation process is known, translation is less well understood. We studied the transcription process by building a convolutional neural network to predict the exact positions of the transcription starts sites. The trained models were then interpreted, which lead to the finding that the area directly around the TSS site is most decisive factor for determining whether a particular base is a TSS, which to best of our knowledge is not reported in literature.

Acknowledgements

The experimental work was carried out at the European Bioinformatics Institute (EMBL-EBI) in Hinxton, United Kingdom. I especially would like to thank Dr. Daniel Zerbino for making the internship possible and pointing me to key resources and providing useful feedback and discussion. I would further like to thank my supervisors at the Leiden Academic Centre for Drug Research and Leiden Institute for Advanced Computer Science Dr. Gerard van Westen and Dr. Wojtek Kowalczyk for reviewing the manuscript. Lastly, I would like to thank the ENCODE-Dream community for providing the data for the transcription factor binding experiment and the FANTOM consortium for publishing the CAGE data.

Contents

1 Introduction **6**

1.1 Motivation 6

1.2 Research objectives and contributions 7

2 Background **8**

2.1 Gene Expression and Regulation 8

2.2 Next-Generation Sequencing 9

2.3 Neural Networks 12

2.3.1 Feed-Forward Neural Networks 12

2.3.2 Stochastic Gradient Descent 14

2.3.3 Backpropagation 15

2.3.4 Regularization 16

3 Methods **18**

3.1 Convolutional Neural Networks 18

3.1.1 Convolution Layer 18

3.1.2 Pooling Layer 20

3.1.3 State of the Art Convolutional Neural Networks 22

3.1.4 Implementation 23

3.2 Interpreting Deep Models 23

3.2.1 Activation Maximization 23

3.2.2 Perturbation based approach 24

3.2.3 Gradient based methods 24

4 Related Work **28**

5 Experiment: Transcription Factor Binding Site Prediction **30**

5.1 Data Description 31

5.2 Prediction Task 34

5.3 Training and Validation 34

5.4 Feature Engineering 35

5.5 Network Architecture 35

5.6 Implementation 38

5.7 Results 38

5.8 Discussion 43

6 Experiment: Transcription Start Site Prediction **44**

6.1 Approach 44

6.2 Results 47

6.3 Discussion 51

7 Summary and Conclusion **52**

1 Introduction

1.1 Motivation

The genetic code that contains the blueprint for the development and functioning of cells is stored in the DNA. The DNA is a long polymer chain that contain four nucleotides: adenine (A), cytosine (C), guanine (G) and thymine (T). Specific regions in the DNA called genes are transcribed into messenger RNA (mRNA), which has uracil (U) in stead of thymine (T). The mRNAs are eventually translated into proteins which perform various functions within the cell. The Human Genome Project started in the early 1990s as an international effort to sequence all of nucleotide base pairs that make up the human genome and identify all the genes. Humans were found to have around 20.000 genes, which is far less than expected before the start of the Human Genome Project. Furthermore, less than 3% of the genome code for a protein. The vast majority of the genome is comprised of noncoding DNA sequences. The noncoding regions nonetheless play an important role in transcription by regulating gene expression.

Cis-regulatory elements (CREs) are regions of the DNA that are important in controlling which genes are expressed depending on cellular state and environment. With the advent of Next-Generation Sequencing (NGS), sequencing costs as well as sequencing time have gone down dramatically. The cost to sequence a human genome has gone down from USD $100 million in 2001 during the human genome project to approximately USD $1000 today. Additionally, current state of the art NGS technologies allow researchers to study not only the genome, but also the transcriptome (RNA transcripts) and epigenome (chemical changes to the DNA and the histone proteins that do not change the sequence) of virtually any organism.

Several public data respositories exists where a vast amount of NGS data has been collected by big consortia. The ENCyclopedia of DNA Elements (ENCODE) project [21] aims to identify all functional elements in the genome and contains various NGS datasets such as protein-dna interaction data in the form of ChIP-seq experiments for distinct cell types. The FANTOM5 (Functional Annotation of the Mammalian Genome) project [22] tries to build transcription regulatory models for all human primary cell types. The dataset is based on transcription start site locations.

These datasets allow researchers to find the novel motifs (short sequence patterns) and epigenomic traits that drive gene expression as well as build predictive models to annotate the genome and identify regulatory elements. Identification of these elements could help future studies and our understanding of gene regulation which could lead to improved medicines and treatment of genetic disorders such as cancers and auto immune diseases.

1.2 Research objectives and contributions

Recently, deep convolutional neural networks (CNNs) have achieved state of the art results on many machine vision benchmarks. These networks generate powerful representations from labeled training data without need the need for manual feature engineering.

In this work two experiments were conducted. In the first experiment we built convolutional neural network models to predict the location of transcription factor binding sites in the genome for use in the ENCODE-Dream transcription factor binding challenge. We did extensive feature engineering as well as a search for optimal architectures for learning. In the second experiment, CNNs were used to build predictive models of the exact location of transcription initiation (transcription start site or TSS) from experimental data. We then interpreted these models to get a better understanding of the transcription process.

The main contributions of this work are: 1) An investigation of various architectures and parameters of convolutional neural networks to integrate various NGS data for transcription factor binding site prediction. 2) By interpreting the convolutional neural network we found strong evidence that the region directly surrounding the TSS site is the most important factor in determining the presence of a TSS as opposed to the TATA box described in literature. 3) Publishing the source code for applying convolutional neural networks to transcription factor binding site prediction, which can be modified to be used on any genomics problem.

The thesis is structured as follows. Section 2 provide an introduction to neural networks and regulatory genomics. Next, Sections 3 and 4 describe convolutional neural networks and their recent applications to genomics. Sections 5 and 6 present and discuss the results of the transcription factor binding and transcription start site experiments respectively. Lastly, we summarize the findings of this work and provide directions for future work in Section 7.

2 Background

2.1 Gene Expression and Regulation

The genetic information used in the growth, development, functioning and reproduction of an organism is encoded into the DNA. The DNA is a long molecule which consists of two complementary biopolymer strands consisting of the nucleotides adenine (A), cytosine (C), guanine (G) and thymine (T). The nucleotides of both strands are bound together with hydrogen bonds between A and T and C and G, forming the double-stranded DNA. These two strands are wound around each other to form a double helix. The asymmetric ends of the DNA strand are identified as either the 3' and 5' end, based on the R group at the end of the sugar molecule. The chromosomal DNA is located in the cell nucleus, where a segment of DNA is compactly wrapped around eight histone proteins each forming a nucleosome. The DNA-protein complex is called chromatin (see Figure 1).

The protein coding genes are parts of the genome that encode function, i.e. they code for a particular protein. The information of genes is copied onto a RNA molecule in a process called transcription. Figure 2 gives a schematic overview this process. In general terms, the following steps are performed during transcription: First the RNA polymerase enzyme and general transcription factor proteins bind to the promoter region of the DNA. Next, the RNA polymerase separates the two strands by breaking the hydrogen bonds of the double helix. One of the strands of the DNA acts as the template strand. Bases on the template strand are read in the 3' to 5' direction one base at a time. At the same time the polymerase builds a complementary RNA chain in the 5' to 3' direction. Sequences called terminators signal the completion of the RNA transcript. In eukaryotic cells (cells with a nucleus enclosed within membranes, e.g. animal or plant cells) the primary transcript undergoes post-transcriptional modifications to turn it into mature RNA. If the RNA codes for a protein, it is called messenger RNA, which eventually is translated to a protein in a ribosome.

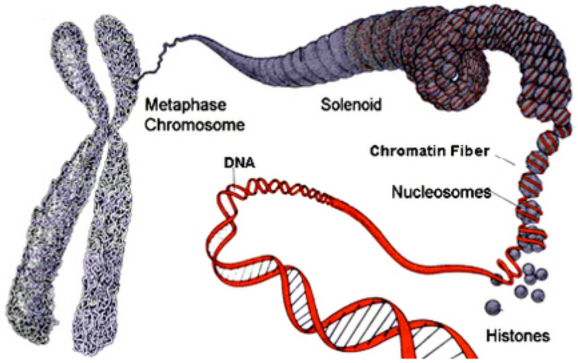


Figure 1: Hierarchical represention of the chromatin structure. Source: O'Sullivan lab at UPCI [2].

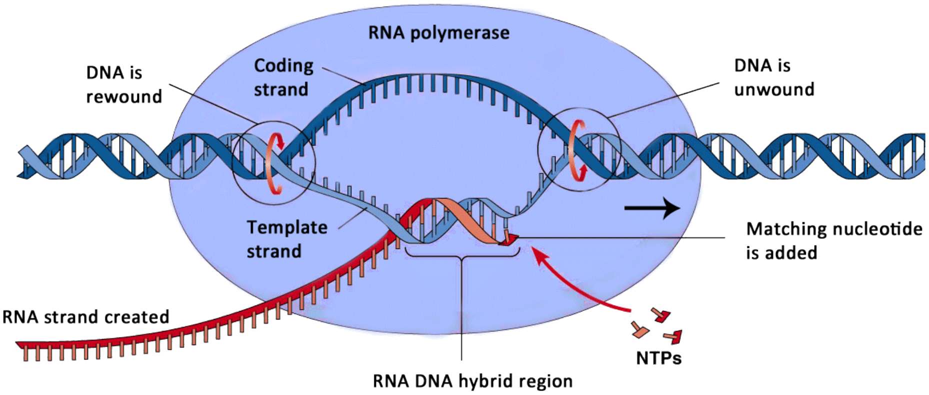


Figure 2: Schematic figure of the transcription process. Source: BIOL 2060, Memorial Univeristy [7].

The location where the transcription of a gene starts is called the transcription start site. Approximately one fifth of the human promoters contain a TATA box 25-25 bases upstream (in the 3' direction of the template strand) of the transcription start site. The TATA box is a so called *cis regulatory element* (CRE), a non-coding DNA sequence which regulates the transcription of nearby genes. Other CREs are *enhancer* or *insulator* sequences. These are short regions in the DNA that can be bound by transcription factor proteins to increase or decrease the likelihood of the transcription of a gene will occur. These transcription factors are encoded in different genes called *trans regulatory elements*. CREs play an essential role in the expression of genes in a cell and therefore have a large impact on the development and functioning of the cell. Changes to these regions could lead to difference in phenotype (e.g. disease). In reality, the exact mechanisms that underly transcription and gene expression are still unknown. Several factors contribute to regulatory events, such as the DNA methylation (addition of methyl groups to the DNA), modification to the histone proteins, chromatin accessibility, transcription factor binding and DNA shape.

2.2 Next-Generation Sequencing

With the advent of next-generation sequencing technologies researchers can study these regulatory events and build predictive models. ChIP-Seq (chromatin immunoprecipitation followed by sequencing) measures protein-DNA binding activity across the genome as well as molecular marks on the histones [35], which can be used to identify transcription factor binding sites. ChIP is a process to enrich certain DNA sequences that are bound by particular proteins. The ChIP process consists of the following steps.

1. Crosslinking the proteins to the DNA
2. Fragmenting the DNA strand by sonification to fragments of 0.2 to 1kb.
3. Linking the target protein with bead attached antibodies specific to that protein.
4. Precipitating or capturing the antibody-protein complex using beads
5. Unlinking the protein and purifying the DNA

At a high level, the data analysis pipeline for a researcher working with ChIP-Seq data is as follows: The ChIP process creates a large number of small reads (small fragments of DNA). These reads are then aligned to a reference genome, which is a complete assembled sequence of a species. These aligned reads can then be converted into a format which can be read by a genome visualization tool like the Integrative Genomics Viewer (IGV) [53] as seen in Figure 3. Usually, there will also be an untreated (no antibodies for that transcription factor) control data set. Regions that are relatively enriched for that transcription factor, known as peaks, can then be found using a peak finder such as MACS [73].

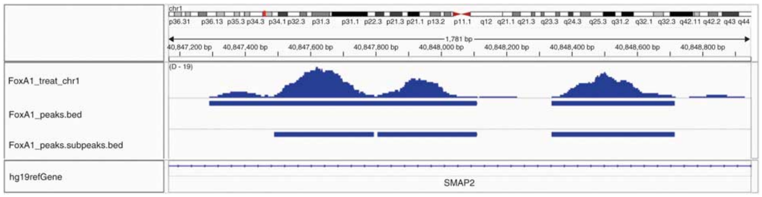

chr1
p36.31 p36.13 p35.3 p34.3 p34.1 p32.3 p31.3 p31.1 p22.3 p21.3 p21.1 p13.2 p11.1 q12 q21.1 q21.3 q23.3 q24.3 q25.3 q31.2 q32.1 q32.3 q42.11 q42.2 q43 q44
1,781 bp
40,847,200 bp 40,847,400 bp 40,847,600 bp 40,847,800 bp 40,848,000 bp 40,848,200 bp 40,848,400 bp 40,848,600 bp 40,848,800 bp
(D - 19)
FoxA1_treat_chr1
FoxA1_peaks.bed
FoxA1_peaks.subpeaks.bed
hg19refGene
SMAP2

Figure 3: Processed ChIP-Seq data visualized in the Integrative Genomics Viewer. Here we can see both the ChIP-Seq signal for the FoxA1 transcription factor as well as the detected peaks, which are in the vicinity of the SMAP2 gene.

Regions of the chromatin that are open, i.e. free of nucleosomes can be found using DNase-Seq (DNase I hypersensitive sites) [61], FAIRE-Seq [26], MNase-Seq [23] and ATAC-Seq (Assay for Transposase-Accessible Chromatin) [16]. Finding these regions is important as the DNA must be accessible for transcription to take place.

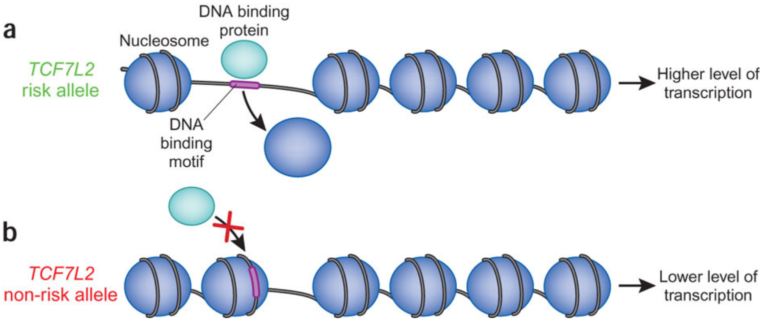


Figure 4: Open chromatin is accessible for transcription factor proteins to bind onto the DNA. Source: Open chromatin and diabetes risk, Groop. [29].

The expression levels of genes can be observed from RNA-Seq analysis [69]. Cap analysis gene expression (CAGE) [56, 40] is a technology which allows for mapping the transcription start sites at single nucleotide resolution. In short, small fragments from the the start of the mRNA are captured, reverse-transcribed to complementary DNA (cDNA) and then amplified using PCR producing short sequences (tags) of 20-27 nucleotides in length. These tags can then be aligned to the reference genome and counted.

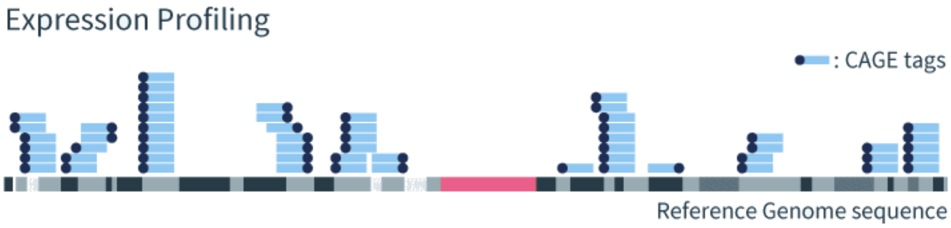


Figure 5: CAGE tags aligned to the reference genome. Each tag corresponds to a mRNA TSS. Source: Kabushiki Kaisha DNAFORM [1].

3D interactions of the chromatin can be captured by Hi-C ((high-throughput chromosome conformation capture)) [12]. Several openly accessible NGS databases have been created by different consortia.

Database	Description
Encyclopedia of DNA Elements (ENCODE) [21]	Comprehensive database of protein-DNA binding data (ChIP-Seq), open chromatin (DNase-Seq, Faire-Seq) and DNA-interactions (Hi-C, 5C) for several cell types.
Functional Annotation of the Mammalian Genome (FANTOM5) [22]	Transcription State Sites for close to 1000 human cell types (CAGE).
Roadmap Epigenomics [14]	Epigenomics database of DNA methylation, histone marks, open chromatin and RNA transcripts (RNA-Seq).
JASPAR [54]	Database of transcription factor binding motifs.

Table 1: Databases with Next-Generation Sequencing data for use in building predictive models for cis-acting regulatory elements.

2.3 Neural Networks

Machine Learning is the field of Computer Science concerned with developing algorithms that allow computers to learn to do specific tasks without being specifically programmed. These tasks are learned from historical data or examples, where the algorithm is 'trained' to model the underlying process and make predictions.

2.3.1 Feed-Forward Neural Networks

A neural network is a computational model approximating a (natural) phenomenon, such as housing prices in a particular neighborhood or the binding of transcription factors to their respective binding sites. More specifically, the neural network produces a predicted output given a number of features describing a particular input. The goal of the neural network is to produce outputs that are close to the observed values in nature. An input $x \in X$ may represent the features of a particular sample for which we want to make predictions, this can be a genomic sequence or the pixel values of an image for example. If the target variable Y is restricted to a set of classes (e.g. binding and non-binding) the problem is a *classification* task, otherwise, if Y takes values from the real numbers, the problem is a *regression* task [15].
A fully connected feed-forward neural network can be represented as a directed layered graph, where the input in one layer is connected to all the outputs in the next layer. Nodes in the intermediary layers are called hidden nodes or *neurons*. This architecture of multiple layers allows neural networks to represent complex interactions in the data [43].
The connections between layers can be represented by a matrix of weights $\mathbf{W} \in \mathbb{R}^{m \times n}$ where m is the number of output nodes and n is the number of input nodes, in particular w_{ij} is the connection weight from input node i to node j. Additionally, the non-input layer nodes will also have a bias term b. The networks adapt to a particular task by modifying the trainable parameters of the network; the connection strengths between nodes and the bias terms of the nodes. A central property of neural networks is that the function is differentiable. This implies that for each parameter we are able to calculate how the output changes to small perturbations of the parameter. Using this gradient information, we can update the trainable parameters by making small steps in order to minimize a distance function between the outputs produced by the network and the expected values obtained from training data.
Given an input, the network performs inference by 'feeding the input forward through the network'. For each incoming connection for a particular node, the input is multiplied by the weight on that incoming edge. All these values are summed up after which the bias term is added. More formally, the representation at layer k, $\mathbf{h_k}$ is computed by the affine transform $\mathbf{h_k} = \sum_{\mathbf{W_k h_{k-1}}} + \mathbf{b_k}$, where $\mathbf{W_k}$ is the matrix representing the weights between layer $k-1$ and k. The output of a particular node may then pass through a non-linear function. In the early days of neural networks, the sigmoid function was used as the hidden layer activation function.

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

The issue of these units is that training can be slowed down considerably when it gets stuck in the saturated part of the input (which means its value will be near the 0 or 1). More recently, rectifiers have been very successful for training deep networks [70]. In particular, the *rectifier linear unit* or ReLU function is defined as

$$ReLU(x) = \max(0, x) \tag{2}$$

By removing the negative part of the input, the representation becomes more sparse. Each next layer changes the representation of the input such that the representation realized at the last hidden layer become linearly separable in the case of classification or able to be fitted by a linear function in the case of regression.
The activation function suitable for the output layer nodes depends on the prediction task. The sigmoid function squashes the output between 0 and 1 and is used in classification tasks to represent a probability of the input being of a instance of a particular class. In the case of multiple classes, where each class will be represented by one node, the softmax function is used. The softmax function takes an n-dimensional vector of real numbers (obtained from the n classes) and squashes the vector into numbers between 0 and 1, summing to 1.

$$\sigma(\mathbf{x}) = \frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}}, \text{ for each class j} \tag{3}$$

Regression tasks do not require a non-linear activation at the final output.
In order for the network to learn a particular function and make predictions the weights of the network are trained to minimize an objective function. This objective function is a function of the trainable parameters of the network (the weights and biases) and consists of a loss function, which returns a distance between the predictions and the expected outcomes, and possible regularization terms which will be described in more detail below. As the network contains millions of parameters and the loss function is in general non-convex, the weights are optimized using a first order method to find a local minimum of the objective function.
Let $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ be the inputs 1, 2, ..., n to the output layer, let y be the expected output and let the output of the network (after applying the activation function) be $\sigma(\sum_{i=1}^{n} w_i x_i + b) = z$, where σ represents the activation function. Depending on the prediction task, we can formulate loss functions as below. Note that these are the losses for one example, in order to calculate the loss for a training batch, the mean of over all the errors on those examples is taken.
For regression problems, the loss function commonly used is the squared distance.

$$C = (y - z)^2 \tag{4}$$

Depending on the distribution of the labels using a mean squared error for classification could result in poor learning due to the derivative having saturated regions. For classification problems, the cross entropy loss function is often used.

$$C = \sum_{i=1}^{m} y_j \ln z_j \tag{5}$$

Where m is the number of classes. We can interpret the output of each training instance as probabilities, that is, the one-hot encoding from the training data and the distribution produced by the model. The cross entropy measures the number of bits needed to encode an instance when using a coding scheme optimized for the model's probability distribution instead of the 'true' probability distribution of the data. The closer the model's distribution is to the real distribution, the fewer bits are necessary to encode the training instance.
A special case of cross entropy for two classes is the binary cross entropy loss function defined as below.

$$C = -(y \ln z + (1 - y) \ln (1 - z)), z \in [0, 1], y \in \{0, 1\} \tag{6}$$

In particular, it has the properties that it is always positive, and close to zero if the output produced by the network is close to the expected output. Furthermore, the derivate of the error function with respect to the weights has the nice closed form of Equation 7. Here, the magnitude of the derivative used to update the weights is directly proportional to the error in the prediction.

$$\frac{\partial C}{\partial w_i} = (\frac{y}{z} - \frac{1-y}{1-z})z'x_i = \frac{z'x_i(z-y)}{z(1-z)} = x_i(z-y) \tag{7}$$

Of course, it is possible for a network to predict several related tasks simultaneously, known as *multi-task* training. In this way, training data can be shared between tasks, which can lead to more general representations and improved prediction performance. For example, in drug discovery, the task predicting biological activity of compounds on a particular protein is related to the task of predicting biological activity on compounds on another protein. For a given compound, a joint model of how that model would interact with several (similar) proteins could be constructed. In this way, knowledge of how compounds interact on a specific protein can be transferred and applied to another protein.

The vast improvements in predictive accuracy of neural networks in the last few years has been accomplished due to better datasets being available for training (e.g. ImageNet [25]), improvements in processing capabilities, specialized libraries being developed and algorithmic improvements, which allow researchers to train increasingly deep models.

2.3.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a well known first-order method for optimizing neural networks. The idea is that by adjusting the weights by taking small steps in the opposite direction of $\nabla Q(\mathbf{w})$, where Q is the function to be optimized or objective function and where ∇ is the vector of partial derivatives of Q known as the gradient. This is done by taking a fixed size batch of training examples, computing the derivatives with respect to all the trainable weights and then updating each weight according to $w := w - \eta \nabla Q(w)$. Here η is the learn rate that controls the step size of the gradient descent.

Vanilla SGD has problems optimizing in regions of the input where the surface curves much more steeply in one coordinate than another. In such regions, SGD will zig-zag along the steepest dimension. By adding a momentum term, the oscillations in such situations can be dampened and the convergence speed will be faster [51]. The update to w at time step t is explained in Equation 10, where γ is the momentum term controlling the importance of past updates.

$$v_t = \gamma v_{t-1} + \eta \nabla Q(w) \tag{8}$$

$$w = w - v_t \tag{9}$$

Nesterov accelerated gradient (NAG) [49] improves on the momentum method by computing the gradient at the expected next position. Since the momentum term γv_{t-1} is used to update the parameters w, the next position can be approximated by $w - \gamma v_{t-1}$.

$$v_t = \gamma v_{t-1} + \eta \nabla Q(w - \gamma v_{t-1}) \tag{10}$$

$$w = w - v_t \tag{11}$$

Another commonly used first-order stochastic optimization algorithm is Adam, proposed by Kigma et al. [39], which improves on the vanilla stochastic gradient descent by dynamically adjusting the updates to the parameters by maintaining estimates of the first and second moment of the gradient.

Algorithm 1 ADAM algorithm. It minimizes (maximizes) $f(\theta)$ using estimates of the first and second moment of the gradient. The parameter α controls the step size, β_1 and β_2 control the exponential decay of the averages of the first and second moment respectively.

```
procedure ADAM ($\alpha, \beta_1 \in [0,1), \beta_2\ in[0,1), f : \mathbb{R}^n \to \mathbb{R}, \theta \in \mathbb{R}^n$)
initialization:
    $m_0 \leftarrow 0$ // estimate of first moment
    $v_0 \leftarrow 0$ // estimate of second moment
    $t \leftarrow 0$ // time step
loop:
    while $\theta_t$ not converged do
        $t \leftarrow t + 1.$
        $g_t \leftarrow \nabla f(\theta_{t-1})$. // compute gradient
        $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ // update first moment estimate
        $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ // update second moment estimate
        $\hat{m_t} \leftarrow m_t/(1 - \beta_1)$ // bias correction due to initialization
        $\hat{v_t} \leftarrow v_t/(1 - \beta_2)$ // bias correction due to initialization
        $\theta_t = \theta_{t-1} - \alpha * \hat{m_t}/(\sqrt{\hat{v_t}} + \epsilon)$
```

By keeping an estimate of the first moment, we smoothen the gradient descent process. The estimate of the second moment is an estimate of the uncertainty. When the variance is high, the step size will be small and conversely if the estimate of the second moment is low, the step size will be larger.

2.3.3 Backpropagation

In order to update the weights to improve the predictive performance of the network, the gradient consisting of all partial derivatives of the error function with respect to the weights needs to be calculated. For a given weight, w_{ij}, which corresponds to the connection strength of hidden node i to hidden node j, the partial derivative is calculated using the chain rule of calculus as below.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \delta_j \frac{\partial}{\partial w_{ij}}(\sum_l w_{lj} a_l + b_j) = \delta_j a_i \tag{12}$$

Here, o_j is the output of the node j before activation and a_i is the output of node i after activation using activation function σ. The activations a are computed during the forward pass and are propagated from input to output. The errors or deltas δ for each node are propagated 'backwards' starting from the error obtained from the objective function down to inputs. The rule to compute these delta's can be again be calculated using the chain rule.

$$
\begin{aligned}
\delta_j = \frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial o_j} = \frac{\partial E}{\partial a_j}\sigma'(o_j) \qquad (13)\\
= \sum_k \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial a_j}\sigma'(o_j) \qquad (14)\\
= \sum_k \delta_k \frac{\partial o_k}{\partial a_j}\sigma'(o_j) \qquad (15)\\
= \sum_k \delta_k w_{jk}\sigma'(o_j) \qquad (16)
\end{aligned}
$$

Hence, computation of the gradient is an efficient process which is similar in terms of computational complexity to computing the forward pass (inference step).

2.3.4 Regularization

Larger networks can learn complex relationships in the data, but they are also prone to overfitting the training data (i.e. fitting to noise in the data or simply memorizing the train set). In order to generalize to new examples, this problem needs to be addressed. By adding a penalty function as a function of the weights to the optimization algorithm, poor representations that do not generalize well can be punished. The objective function Q then consists of the cost function C and regularization term(s). L2 regularization penalizes the squared magnitude of the weights. Let $\mathbf{w}$ be the weights of the network. The objective function Q then becomes $C + \lambda\|\mathbf{w}\|_2^2$, where $\lambda \geq 0$ is the parameter controlling the impact of large weights on the objective functions. In particular, the larger the value for λ, the more the network is penalized for large parameter values.

Another simple to implement, effective and often-used method is dropout [63]. It attempts to prevent the co-adaptation of neuron activations, where several neurons together learn a specific feature. During the training of the network, for each separate example in the training batch, the activations of randomly selected neurons is set to zero. This results in more general, sparse and less correlated representations and therefore improved performance on held-out training sets [19, 63].

Another issue of deep networks is the exploding or vanishing gradient problem, where gradient information is unable to propagate through many layers. The ReLU non-linearities address this by creating more sparse representations. Since only a subset of the nodes will be active for a given example and the computations will be linear on this subset, gradient information will be propagated down the network more easily [28]. Moreover, since only a subset of neurons will be active, adding ReLUs also improves the speed of inference and training.

Related to the learning problem caused by vanishing or exploding gradients in deep neural network architectures, is the phenomenon that the distributions of each layer's inputs may change during training. As each layer is affected by the parameters in all previous layers, small changes to the network parameters have an amplifying effect further down the network. The change of distributions of internal nodes of a deep neural network is referred to as *Internal Covariance Shift*. Batch normalization proposed by Ioffe et al. [33] is a way to reduce this phenomenon. The idea is to normalize or whiten the activations of each feature separately. Consider a layer with n inputs $x^1, x^2, ..., x^n$. The normalized value for the i-th input is calculated as

$$\hat{x}^i = \frac{x^i - E(x^i)}{\sqrt{Var(x^i)}} \tag{17}$$

For computational efficiency, the batch mean and batch variance are used to translate and scale the input. The batch normalization procedure for a particular layer can be summarized as in Algorithm 2. The shift and scale operation by parameters γ and β is necessary to ensure the network retains the representational power, that is, the ability to approximate certain functions, by allowing the batch normalization transform to represent the identity function.

Algorithm 2 Batch Normalization transformation. Given m training examples in a mini-batch, each feature x with training examples $x_1, x_2, ..., x_m$ is scaled and shifted to y.

```
procedure BN $(x_1, x_2, ..., x_m)$
    $\mu \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // batch mean
    $\sigma \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$ // batch variance
    $\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma + \epsilon}}$ // normalize
    $y_i = \gamma\hat{x}_i + \beta$ // scale and translate
```

During training of the network, the batch normalized inputs are replaced by their respective transformations $y = BN(x)$. The model parameters to be optimized are the weights and biases of the initial model as well as the scaling and translation parameters of the batch normalization. Once the model has been trained, the batch-normalization transform is made deterministic by making use of population statistics for the mean and variance. More specifically, for each feature, we take the sample mean over the batch means and the unbiased sample variance of the batch variances. Afterwards, the batch normalization operation can be replaced by a linear transform.

The advantage of using batch normalization is that it enables higher learning rates, since normalizing prevents small changes in the parameters to have an amplifying effect further down the network. Furthermore, it makes the network robust to scaling of a layer's parameters. High learning rates may increase the scale of the weights which may lead exploding gradients during backpropagation. Now for a scaling factor a, we have that $BN(wx) = BN((aw)x)$, since

Using batch normalization however, the error propagation will be unaffected by a scale of the parameters. Given the affine transform $wx+b$, we have that $BN(wx+b) = BN((aw)x+b)$ for scaling factor a. This results in the following observations for error propagation and gradients of weights respectively.

$$\frac{\partial BN((aw)x + b)}{\partial x} = \frac{\partial BN(wx + b)}{\partial x} \tag{18}$$

$$\frac{\partial BN((aw)x + b)}{\partial aw} = \frac{1}{a}\frac{\partial BN(wx + b)}{\partial w} \tag{19}$$

That is, the error propagation is unaffected by the scaling of the weight parameters and the size of the gradients is inversely proportional to the size of the weights, hence parameter growth is stabilized.

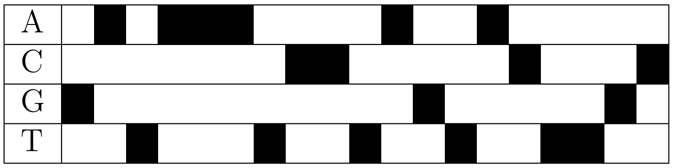
Furthermore, batch normalization also acts as a regularizer. As mini-batches consist of different examples, the activations for a particular training example will vary across mini-batches. In practice, Ioffe et al. [33] found batch normalization leading to better generalization and reducing the need for dropout.

3 Methods

3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural networks that can process types of data that have a grid-like topology, that is, that have statistical dependencies between neighboring elements. An example of such data are images, which are a 2-D grid of pixels. CNNs have been very successful in machine vision tasks achieving state of the art performance in various image and video recognition benchmarks.

Similarly, the DNA can be thought of as a 1-D grid of nucleotides. More specifically, a n length sequence of DNA can be represented as a $4 \times n$ matrix, where each nucleotide is represented by a vector of length 4. An example encoding is given in Figure 6.



A		■		■	■	■					■			■					
C								■	■						■				■
G	■											■						■	
T			■				■			■			■			■	■		

Figure 6: One-hot encoding of the sequence GATAAATCCTAGTACTTGC

This encoding can be compared to an image where instead of being a 2-D grid having three channels R, G and B, the genome is represented by a 1-D signal with four channels A, C, G and T. Additionally, any signal over the genome, such as chromatin accessibility, histone marks or transcription factor binding data can be processed using the convolutional neural network by either adding it as a separate channel or processing it separately. The components of vanilla convolutional neural networks are described below.

3.1.1 Convolution Layer

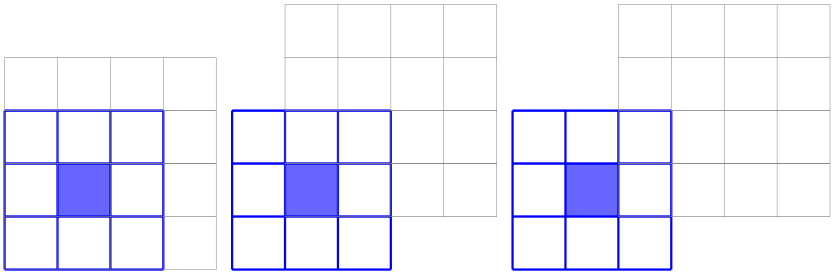
Convolution is a mathematical operation which takes two functions f and g as input and produces a function which is the integral of pointwise multiplication of f and a reversed and shifted version of g. Formally, the convolution operator $*$ is defined as in Equation 20.

$$(f * g)(t) := \int_{-\infty}^{\infty} f(x)g(t-x)dx \tag{20}$$

In convolutional neural networks, the function g is called a *convolutional filter* and acts on the *input* function f. The resulting output is called a *feature map*. In practice, the input will be a multidimensional array such as an image or the one-hot encoded genome. The convolutional filter is a multidimensional array of parameters, consisting of the weights of the connections and the bias. Consequently, the convolution operation is a finite summation over one or multiple axes. More concretely, let I be the input array of N dimensions and K be a kernel, then the output at $\mathbf{i} = (i_1, ..., i_N)$ is calculated as in Equation 21, where $\mathbf{j} = (j_1, ..., j_N)$ is a coordinate of the input.

$$(I * K)(\mathbf{i}) = \sum_{\mathbf{j}} I(\mathbf{j})K(\mathbf{i} - \mathbf{j}) \tag{21}$$

Note that the convolutions at the borders present an edge case, since we may need coordinates which are not part of the input. There are several ways to deal with the border cases, visualized in Figure 7.



(a) valid convolutions (b) same convolutions (c) full convolutions

Figure 7: Visualized are three two-dimensional 3×3 convolutional filters acting on two-dimensional grids (e.g. an image). The center of the kernel is, depicted by the filled in blue square, accumulates information about the neighboring cells inside the 3x3 kernel. By moving the kernel over the image the feature map is created. There are three ways one can deal with the border cases when performing convolutions. *Valid* convolutions only allow each element in the kernel to overlap with the input grid. *Same* convolutions restrict the center of the kernel to overlap with the input grid. In this way, the dimensions of the input and the produced feature map will be the same. Lastly, *full* convolutions only require part of the kernel to overlap with the input. The positions in the kernel that do not overlap with the kernel can be padded with zeros in the input.

In the case of a genome, the input will be represented by a one-hot encoding of the DNA sequence. The convolutional filters act on local parts of the genome as depicted in Figure 9. Note that each nucleotide will be represented by a vector of length 4. Consequently, each connection in Figure 9 contains 4 weight variables. The filter in this case is a matrix with 4 rows and n columns, where n is the width of the filter. The output of the filter is then calculated as the sum of the entries of Hadamard product between the filter matrix K and the relevant portion of the input, i.e. the region of the genome that is connected to that filter, I. More precisely, the output of the i-th filter of length w at position j is calculated as in Equation 22 with the addition of a bias term. Sometimes, the filter K is not flipped like in the equation, in this case we speak of cross-correlation instead of convolution.

$$\sum_{k=0}^{w} \sum_{n=1}^{4} I_{n,k+j} K_{n,w-k} \tag{22}$$

As stated earlier the filter is a $4 \times n$ matrix, where n is the width of the filter. Each filter scans the genome for a certain sequence motif. The filter can be represented graphically as in Figure 8. Note the similarity between a convolutional filter and a position weight matrix which are commonly used in bioinformatics to represent sequence patterns [65, 66].

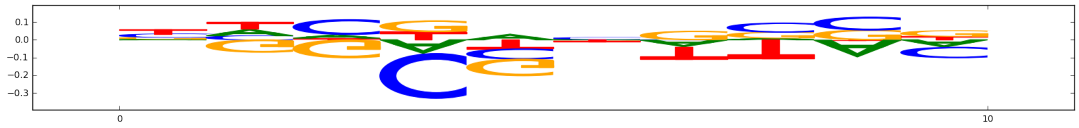


Figure 8: Graphical representation of a convolutional filter for scanning a DNA sequence. Each position in the filter contains four values for each of the bases. Positive values for a particular base indicate that the filter activation will be higher if that base is present at that specific position. Conversely, if a base is present on the position where the filter has negative value for that particular base, the resulting score of the filter will be lower. As such the filter acts as a detector for specific sequence motifs. For example in this visualization, the filter is looking for a TTCG pattern for the first four bases.

In contrast to traditional multilayer neural networks, where higher level neurons are connected to all previous neurons, the convolutional units are only connected to a local region in the input. This concept is referred to as *sparse connectivity* allows the CNN to more efficiently exploit the local correlations. Additionally, adjacent neurons share the same weights. By restricting the model in this way, fewer parameters need to be stored. This reduces memory requirements and computation time and simplifies the learning task. Intuitively, one could view convolution as applying the same local filter using a set stride (for example starting at each base pair).

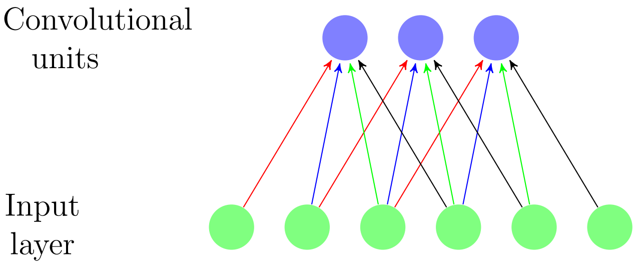


Figure 9: Local connectivity and weight sharing in the convolutional layer. In this case the filters have a width of 4. Similar colors indicate similar weights.

In the next stage of the convolution layer, the output of the convolutional units is passed to a nonlinear function, which is often referred to as the *detector stage* of a convolution neural network.

3.1.2 Pooling Layer

Convolution layers often precede pooling layers. The pooling layers summarizes the output of the previous layer by downsampling the input. For example in a *max-pooling* layer, the input is down sampled by taking the maximum over a fixed length region. Max-pooling has several advantages; it reduces the number of parameters of the model and helps make the representation more invariant to translation of the input. The canonical representation for a convolution and pooling unit is depicted in Figure 10. In practice however, the non-linear detector stage and the pooling layer can be swapped in order to reduce computation time since the $max(ReLU(x)) = ReLU(max(x))$.

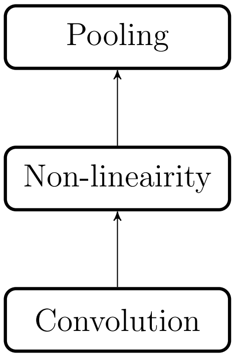


Figure 10: Convolutional unit consisting of convolutions, followed by a non-linear activation function and down sampling stage through pooling.

Convolution/pooling layers can be stacked onto each other. Neurons in higher layers will cover increasingly larger parts of the input (visual field) and generate increasingly higher level feature representations. The convolution / pooling layers are often followed by dense layers with fully connected units. Figure 11 shows an overview of the VGG-16 or OxfordNet architecture [59]. As can be seen, the representations generated by the convolutional layers get increasingly smaller in width and length and greater in terms of depth. Each position in the depth slice of a representation covers an increasingly large portion of the input as each neuron covers multiple input neurons. More formally, each neuron i in layer j covers the outputs of the connected neurons at layer $j-1$, each of which is connected to the layer below until the input layer. As the depth increases, the number of features extracted for that position increases, i.e. the number of filters is increased. Intuitively, we use more features to describe higher level features which cover a larger part of the input.

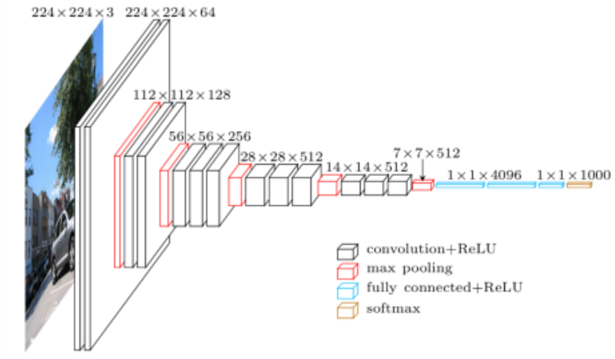
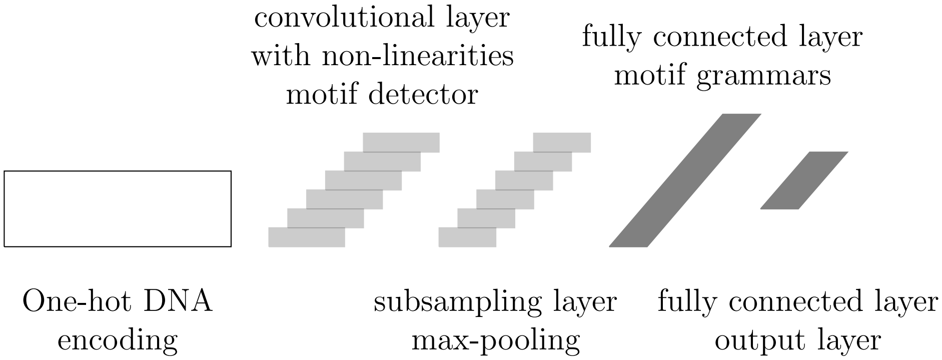


Figure 11: VGG-16 architecture, which is a popular architecture for image recognition. Source: Fossard, University of Toronto [6].

In genomics, lower convolutional layers act as motif detectors. Higher layers take combinations of the output of these motif detectors to learn higher level features such as motif grammars. For example for enhancers the composition of motifs and motif grammars can be fixed or flexible, depending on the enhancer model [62]. A CNN can learn these patterns automatically from data.



When performing backpropagation for convolutional layers, the delta term for a particular weight must be summed up for all expression in which that variable occurs due to weight sharing between feature maps. Similarly, for batch normalization, the same normalization is applied to all locations in a feature map. For max-pooling layers, only the node that produced the maximum activation during the forward pass carries the deltas for lower layers. During the forward pass, we can keep track of the 'winning' unit and propagate the error signal through that node.

3.1.3 State of the Art Convolutional Neural Networks

Deep neural networks are able to build more complex representations of the data and the best performing neural networks on competitions have been increasingly deep. In theory a deeper network should be able to perform at least as well as a shallow network, since the shallow network could just be learned by the deeper network and the next layers could learn the identity function. However, training deeper neural networks has proven challenging. It has been shown that deeper networks suffer from a *degradation* problem, that is, as the network gets deeper by adding more layers the accuracy actually gets worse even though gradients do not vanish or explode. He et al. [30] address this problem by learning residual representations. Let the mapping to be learned by a stack of layers be $H(\mathbf{x})$. Instead of learning this representation directly, the stacked layers in a residual neural network learn the residual mapping $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$. The original mapping is then obtained as $H(\mathbf{x}) = F(\mathbf{x}) + \mathbf{x}$. The reason for doing this is that learning the identity function may in fact for a neural network. Adding the skip connections helps preconditioning the problem, in that, the optimization process can start close to the identity mapping, which ensures the network performs at least at the level of a shallow network. In the case that the dimensions of $F(\mathbf{x})$ and $\mathbf{x}$ do not match, 1x1 convolutions could be used to match the dimensions. Using this architecture He et al. were able to train a 152 layer neural network on the ImageNet dataset winning first place in the 2015 ILSVRC competition.

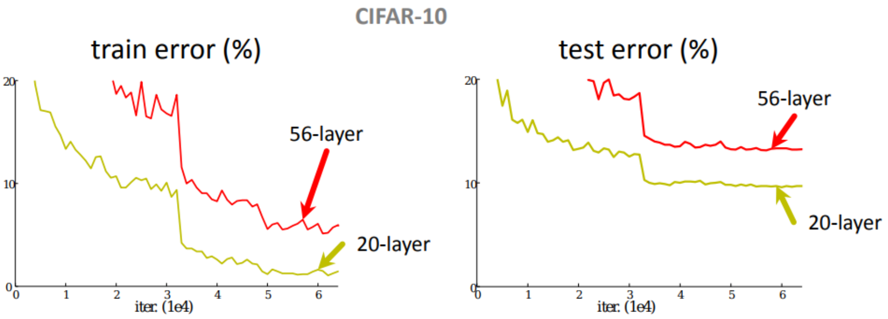


Figure 12: Degradation problem in vanilla convolutional neural networks. Adding more layers leads to an increased error on both the train and test set on the CIFAR-10 benchmark [41]. Source: He et al. Microsoft Research Asia [30].

3.1.4 Implementation

Specialized libraries for neural network models with built in routines for performing convolution operations, performing symbolic differentiation and optimizing the network weights such as TensorFlow, Theano, Torch and Caffe [9, 13, 20, 34] allow researchers to develop models on a high level of abstraction. In particular in the case of TensorFlow, a computation is described in the form of a graph. The nodes on the graph represent operations, which are abstract representations of computation such as addition, matrix multiplication or a non-linearity. *Tensors*, arbitrary dimensional arrays, flow along the edges of the graph. TensorFlow provides the means to execute these computations on one or multiple devices. Graphical Processing Units (GPUs) are often used to train neural networks, since the computations involved can easily be parallelized and therefore GPUs provide significant speedups over general purpose CPUs.

3.2 Interpreting Deep Models

Deep networks have proven to achieve exceptional results on various machine learning tasks by creating high level representations of the data. However, considering these networks are comprised of multiple layers and contain millions of parameters, interpreting these deep networks to gain an understanding what the network has learned is not straightforward.

In the case of genetics, we are particularly interested in finding what sequence motifs are important in certain processes (e.g. the binding of DNA-proteins). Formally, a motif is a fixed-length sequence pattern with biological significance represented by a $4 \times n$ matrix of real numbers, where each column represents a probability distribution over the nucleotides A, C, G and T for that position. For example, a motif could be a certain pattern where transcription factor proteins bind onto. Currently, the standard tool for motif discovery is MEME [11], which uses the Expectation Maximization algorithm [24], where the starting position of a motif first get estimated (E-step) and then the k-length motif is updated (M-step). For large genetics datasets, the more efficient MEME-ChIP is often used [47].

3.2.1 Activation Maximization

The lower layer convolutional filters act as motif detectors. In fact, once the model is trained, we can use these filters as feature extractors to generate representations that can be fed into other machine learning algorithms such as random forests [46]. As each filter is in effect a position weight matrix, a simple approach then would be to visualize the weights of these filters as shown in Figure 8.

A different way of visualizing what a filter has learned is to find inputs that maximize the activation of that particular filter. More specifically, we first train a network $f : X \rightarrow Y$ with with trainable parameters θ. After training the network, the trainable parameters θ of the network are fixed. Now, instead of optimizing the parameters of f by minimizing an error function with respect to θ, the output of a particular neuron is maximized with respect to the input. Let $g_i : X \rightarrow \mathbb{R}$ be the function that outputs the activation for neuron i for a given input. The optimization problem can then be stated as

$$\underset{\mathbf{x}}{\operatorname{argmax}}\, g(\mathbf{x}) - \lambda ||s||_2^2 \tag{23}$$

The term $\lambda||\mathbf{x}||_2^2$ penalizes large values in the sequence representation and is necessary otherwise any input with large weights will produce a large activation. Similar to training the network, we can find an input $\mathbf{x}$ which maximizes activation using gradient ascent. Note that the starting point of $\mathbf{x}$ should be chosen carefully such that the optimization process does not get stuck in a bad local optimum or is biased towards certain solutions.

Simonyan et. al [58] use this process to find the input that maximizes the activation for a particular class of image. With this we can find images which represent the particular class.

Similarly, for DNA sequences, given a trained network we can find inputs sequences that maximize the activation of a particular class (e.g. transcription factor FOX1 bound / unbound). Let $s \in \mathbb{R}^{n \times 4}$ be the encoding of the sequence and $f_C : \mathbb{R}^{n \times 4} \rightarrow \mathbb{R}$ be the score of class c obtained from the CNN. We then find the sequence that maximizes the class score: $\arg\max_s f_C(s) - \lambda||s||_2^2$. A good starting point is an encoding where each nucleotide (A, C, G and T) is equally likely (e.g. starting with each nucleotide having a score of 0.25). Note that this representation is not a strict sequence as seen in the previous section (i.e. a one-hot encoding).

3.2.2 Perturbation based approach

The contributions of particular parts of the input to the prediction can be examined by modifying the input and observing the resulting change in a particular node. For DNA-sequence data the individual bases can be mutated and the effects on the model observed, performing an in-silico mutagenesis. For example, given a model for a particular phenomenon (e.g. transcription factor binding), the effect of mutating base pairs on the binding of transcription factors can be investigated. This can be particularly interesting to investigate the effects of mutations at specific points in the genome associated with a phenotypic change (such as disease), also known as single nucleotide polymorphisms (SNPs). The drawback of using these types of methods for interpreting the model is the computational complexity considering the large number of possible changes that can be made to the input sequence (4^n, where n is the length of the sequence under investigation). Each such change would require a full forward pass.

3.2.3 Gradient based methods

Since the subset of the network that is active (i.e. where the neurons fire) is differentiable, the gradient information can be used to calculate the contributions of the individual nucleotides on the activation of specific nodes. Simonyan et al. [58] propose a method to calculate the contributions individual elements in the input on the score (activation) of a specific class. First, given an input I_0 and the activation of a particular node specific to that class calculated by the affine function, $S_c(I_0) = w_c^T I_0 + b_c$, the contributions of the individual inputs of I_0 will be simply given by the weights w_c. That is, large values of elements of w_c indicate that the corresponding elements from I will have a large influence on the score S_c. A deeper neural network can not be interpreted directly in this way. However, we can calculate the first order Taylor expansion approximating the non-linear score function given by the deep neural network for a particular class S_c in the neighborhood of the image I_0 as follows

$$S_c \approx \left.\frac{\partial S_c}{\partial I}\right|_{I_0} I + S_c(I_0) \tag{24}$$

Here $\left.\frac{\partial S}{\partial I}\right|_{I_0}$ is the derivative of f with respect to input I at point I_0. Similar to the elements in w_c, the magnitude of the elements in the term $\left.\frac{\partial f}{\partial I}\right|_{I_0}$ can be interpreted as the importance of each of the parts of the input.

A disadvantage of using gradient based methods is that neurons that do not fire and hence will have a gradient of zero can still carry information. For example consider the network.

$$f(x_1, x_2) = 1 - ReLU(1 - x_1 - x_2) = 1 - max(0, 1 - x_1 - x_2) \tag{25}$$

Now,

$$y = x_1 + x_2 \text{ when } x_1 + x_2 < 1 \tag{26}$$

$$y = 1 \text{ when } x_1 + x_2 >= 1 \tag{27}$$

Hence, the gradient will be zero in the saturated region of $x_1 + x_2 >= 1$. Therefore, calculating contributions as in the method described above will lead to zero contributions for both inputs in this region of the input.

To deal with this problem a new method for computing the importance of the inputs by computing contributions of those inputs to a difference in activation to a reference input called DeepLIFT was proposed by Shrikumar et al. [57]. The idea is to take a neutral input/example called the reference input, for example the input where each nucleotide is equally likely, and compute the contributions of each of the nucleotides to the difference in the final activation for some example sequence.

In more detail, suppose we have a network $f : X \to [0, 1]$, which maps the input to one real valued output between zero and one, predicting whether a transcription factor binds to the sequence given by the input. The network under the reference input will produce a certain activation as it will given an example input. We would like to find out what the contributions of the individual input bases are to the difference in the activation under the reference and the example input. Assuming the neural network has found a good model of the data, this will give an indication as to which bases are important in the process the network is modeling. The choice of the reference will have great impact on the results and must be chosen to be a 'neutral' example, e.g. the background frequencies for the nucleotides for genetics problems.

Formally, let C_{xy} denote the contribution of node x to node y. Next, let A_n and A_n^0 be the activation of node n in the example input and the reference input respectively. The difference in activation from reference is then defined as

$$\delta_n = A_n - A_n^0 \tag{28}$$

Finally let S_y be the set of nodes that are minimally sufficient (i.e. non-redundant) to compute the activation of y and let O_x be the output neurons of x. The contributions are then defined by the following two properties:

$$\sum_{s \in S_y} C_{sy} = \delta_y \tag{29}$$

$$C_{xy} = \sum_{o \in O_x} \frac{C_{xo}}{\delta_o} C_{oy} \tag{30}$$

The first property is straightforward, the contributions of the neurons from which the activation of node y can be calculated should sum up to the difference in activation. Using the second equations contributions of lower layers can be computed from contributions of higher layers. In the example network f above, given the reference $x_1 = x_2 = 0$, the distance from reference at $x_1 = x_2 = 1$ will be 1 and hence contributions of x_1 and x_2 will be 0.5 for each, even though the gradient equals zero.
The contributions for a neural network can be calculated using the rules below. More specifically, in order to avoid numerical problems when the difference from reference δ_n for a node n is small, multipliers as defined below are calculated.

$$m_{xy} = \frac{C_{xy}}{\delta_x} \tag{31}$$

Chain rule The first rule, which is reminiscent of the chain rule seen in back-propagation, allows for calculating multipliers for multi-layered network. Given a node t, which is in a higher layer than x, but connected to x, we can calculate the multiplier m_{xt} as below using Equation 30.

$$m_{xt} = \frac{1}{\delta_x} \sum_{o \in O_x} \frac{C_{xo}}{\delta_o} C_{ot} = \frac{1}{\delta_x} \sum_{o \in O_x} \frac{m_{xo}\delta_x}{\delta_o} m_{ot}\delta_o = m_{xo}m_{ot} \tag{32}$$

Affine functions For affine functions, such as for convolutions and fully connected layers. The following rule applies. First let I_y denote all the input nodes of node y. Let

$$A_y = \sum_{x \in I_y} w_{xy} A_x + b \tag{33}$$

It can be shown that $m_{xy} = w_{xy}$. Using Equation 28, we have $A_n = \delta_n + A_n^0$.

$$A_y^0 + d_y = \sum_{x \in I_y} w_{xy}(A_x^0 + \delta_x) + b = \sum_{x \in I_y} w_{xy} A_x^0 + b + \sum_{x \in I_y} w_{xy}\delta_x \tag{34}$$

By definition,

$$A_y^0 = \sum_{x \in I_y} w_{xy} A_x^0 + b \tag{35}$$

Substituting $\sum_{x \in I_y} w_{xy} A_x^0 + b$ by A_y^0 in the above equation and canceling out the A_y^0 on both sides gives in the desired result.

$$\delta_y = \sum_{x \in I_y} w_{xy}\delta_x = \sum_{x \in I_y} C_{xy} = \sum_{x \in I_y} m_{xy}\delta_x \implies w_{xy} = m_{xy} \tag{36}$$

Max-pooling A max pool operation can be defined as

$$A_y = \max_{x \in I_y} A_x \tag{37}$$

The multipliers can be computed as

$$m_{xy} = 1\{A_x = Ay\}\frac{\delta_y}{\delta_x} \tag{38}$$

The term $1\{A_x = Ay\}$ equals 1 if the condition inside the braces is true and 0 otherwise. Equation 38 can be easily proven from the definition of contributions.

$$\sum_{x \in I_y} C_{xy} = \sum_{x \in I_y} m_{xy}\delta_x = \sum_{x \in I_y}(1\{A_x = Ay\}\frac{\delta_y}{\delta_x})\delta_x = \sum_{x \in I_y} 1\{A_x = Ay\}\delta_y = \delta_y \tag{39}$$

Non-linearities For non-linearities (ReLU, sigmoid), we can use the following rule.

$$m_{xy} = \frac{\delta_y}{\delta_x} \tag{40}$$

This follows directly from Equation 29.

4 Related Work

Convolutional neural networks have been applied to modeling and studying epigenomic events in several studies. In most studies the architectures used are shallow networks, consisting of one convolution layer followed by a ReLU non-linearity, a max-pool downsampling layer, one dense hidden layer and lastly one output layer [71]. Multi-task networks are also common as the sequence is shared between different prediction tasks and cell tissues. These networks are very flexible in that they can process a variety of NGS data and can be applied to study many different chromatin effects. Basset [36] and DeepBind [10] use the shallow multi-task network to predict chromatin accessibility and transcription factor binding respectively.

A deeper and much wider architecture called DeepSea was used by Zhou et al. to build a multi-task network to predict 919 chromatin features including transcription factor binding, open chromatin and histone modifications using datasets from the ENCODE and Roadmap Epigenomics projects from sequence alone [74]. The network architecture consists of three convolution/pooling layers, one hidden dense layer and lastly one output layer with 919 outputs corresponding to each of the prediction tasks. Quang et. al [52] improve on the performance of DeepSea by using a shallow architecture but adding a bi-directional long short term memory layer in between the pooling and fully connected layers. A long short term memory or LSTM [31] is a recurrent neural network (RNN) layer. RNNs processes data from left to right, keeping track of an internal state that is updated by the inputs. By doing this, the RNN is able to use information of previously seen inputs when making predictions. Recurrent neural networks are also trained using backpropagation, taking into account that a particular output can be influenced by all previous inputs, hence error information is propagated backwards through the time steps. LSTMs consist of blocks, where each block contains a memory cell maintaining the block's state. Updates to and reads from the memory cell's content is regulated by a gating mechanism. Using this architecture an LSTM is able to recall information from arbitrarily many time steps before. The rationale for using the recurrent layer is the claim that it can more efficiently capture motif grammars caused by the spatial arrangement and composition of motifs in vivo than dense layers.

Schreiber et al. successfully apply convolutional neural network to predict 3-D chromatin interactions from sequence and chromatin accessibility given by DNaseI experiments [55]. The interaction data is taken from Hi-C experiments. The model takes as input two genomic regions with their respective sequence and DNaseI signal and maps that to the confidence score that the two regions will interact. The two regions are processed separately. First the DNase and sequence are processed separately, where the sequence is processed by two convolution/pooling stages and the DNase signal is processed by one pooling/convolution unit. Next the resulting representations are combined and undergo more processing by a convoluting/pooling stage. The representations for both regions are then merged and go through a final dense unit, where the distance between the regions is added as a separate input.

Poplin et al. developed a SNP and variant detector from aligned reads, by encoding the reference and read bases and read features as images [50]. These images are then processed by the Inception-V2 architecture [68]. The approach, called DeepVariant, achieved the highest performance in an FDA administered variant calling challenge. This architecture is significantly deeper than the previously mentioned works.

Common approaches to interpret the trained network are to observe the activation of particular filters or outputs of interest. In particular perturbation based approaches are often used to study the effects of single nucleotide variants on various epigenomic events [10, 36, 74]. DeepBind finds motifs from data using the activations of the different convolutional filters of the trained model. It takes all the instances from the test set and for each instance and each convolution filter k it finds the position which has the highest activation score after rectifying passing some pre-set threshold. The sequence corresponding to that position is kept and all such sequences for all the examples in the test set are used to create a position frequency matrix. This PFM can then be visualized as a sequence logo (similar to Figure 8) in the usual way. Using this approach DeepBind has been able to reproduce known motifs for various transcription factors. The activation maximization approach described in the previous section was used to produce a visualization of profiles of histone modification data to identify which histone marks are indicative of a nearby TSS site [60].

5 Experiment: Transcription Factor Binding Site Prediction

Transcription is the process where a particular DNA segment (gene) is copied into messenger-RNA by the enzyme RNA-polymerase. Transcription factors (TFs) are proteins that bind to specific parts in the non-coding part of the genome to promote or repress the transcription and thereby regulate the expression of nearby genes. In July of 2016, The Encyclopedia of DNA Elements (ENCODE) Project [21] together with Dream Challenges launched a competition on Synapse [8] to build predictive models of transcription factor binding across cell types. The models can use the primary DNA sequence, open chromatin (in the form of DNase-Seq data), gene expression (RNA-Seq) and DNA shape in order to predict transcription factor binding sites which are determined from ChIP-Seq experiments. For each of the transcription factors there will be one or more cell types for which the ChIP-seq data is provided as well has one or more held out cell types for which the ChIP-seq data has not yet been released by ENCODE. The held out cell types are divided into leaderboard cell types, for which 10 submissions can be made and final submission cell types, for which only a single submission is allowed.

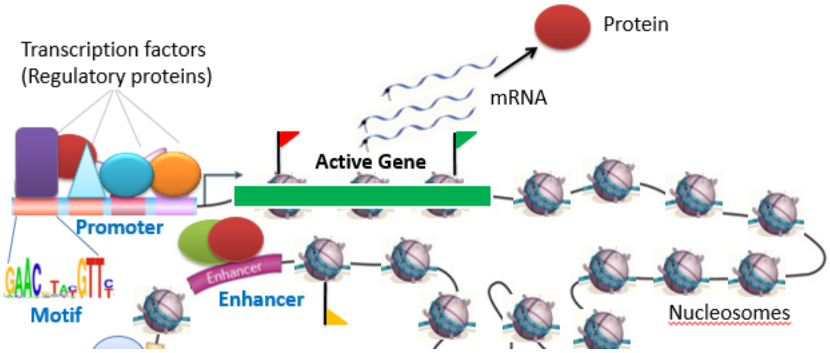


Figure 13: The goal of the challenge is to predict the locations of the binding locations in the genome of specific transcription factor proteins. These TFs regulate the expression of nearby genes. Source: Encode-Dream Transcription Factor Binding Challenge [4]

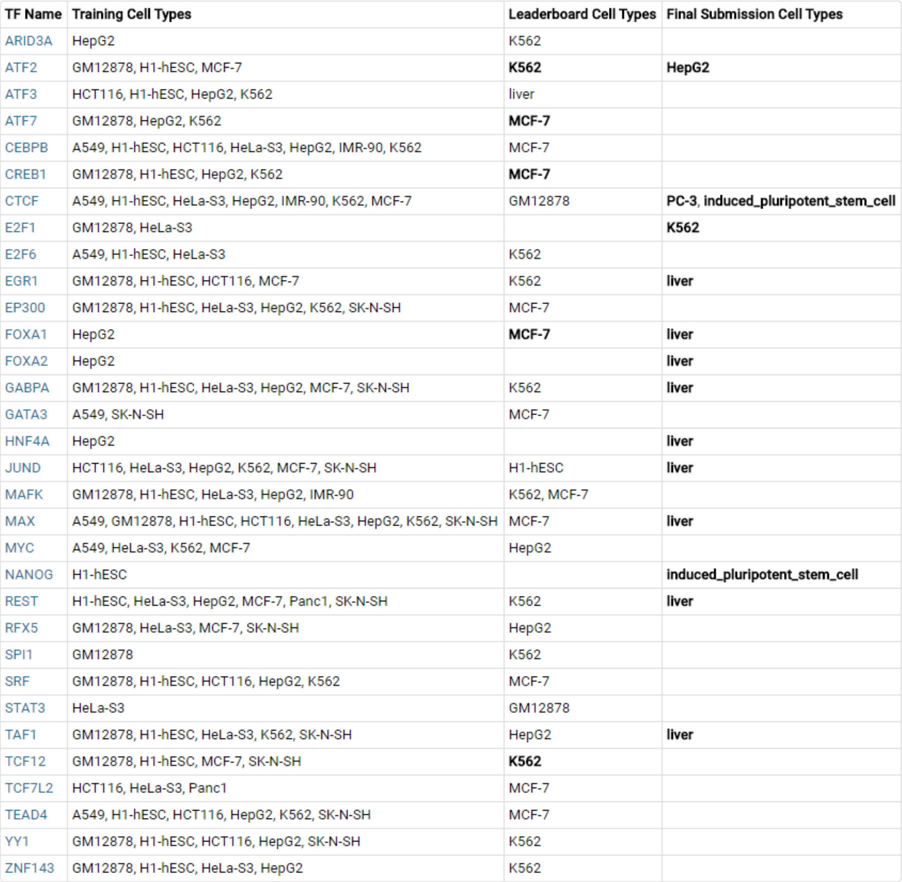
The prediction task is structured as a classification problem. The genome is divided into overlapping bins of 200 base pair(bp) length with a stride of 50, where each bin is classified as being either unbound by the transcription factor, having an ambiguous peak or a conservative peak. An example is given in the Table 2. For the held out train sets the model then have to assign a probability to each bin being a conservative peak.

Chromosome	Start	Stop	Label
chr10	10000	10200	U
chr10	10050	10250	U
chr10	10100	10300	A
chr10	10150	10350	A
chr10	10200	10400	C
chr10	10250	10450	A
chr10	10300	10500	A
chr10	10350	10550	A

Table 2: The genome is divided into regions of 200 bp length. For each bin a prediction the model needs to predict whether the transcription factor binds to this part of the DNA as determined by ChIP-seq experiments. The label U means the TF does not bind at this location, A is an ambiguous binding site and is counted as unbounded (U). Lastly, C means a conservative peak region, that is, the transcription factor is found to be binding at this location.

5.1 Data Description

As mentioned in the previous section, experimental data is used for both the input of the predictions (DNA sequence, chromatin accessibility and gene expression) as well as the output target for the predictions (The transcription factor binding sites). In total there are 32 transcription factors and 14 cell types. Please refer to the overview in Figure 14 In the sections below a more detailed description of the data sources is given. The held out cell types of the leaderboard round only consist of chromosomes 1, 8 and 21. The training cell types contain all chromosomes except for the Y-chromosome and the held out chromosomes. The final cell types consist of all chromosomes except for chromosome Y. The total size of all the provided data is around 2.2TB uncompressed.



TF Name	Training Cell Types	Leaderboard Cell Types	Final Submission Cell Types
ARID3A	HepG2	K562	
ATF2	GM12878, H1-hESC, MCF-7	**K562**	**HepG2**
ATF3	HCT116, H1-hESC, HepG2, K562	liver	
ATF7	GM12878, HepG2, K562	**MCF-7**	
CEBPB	A549, H1-hESC, HCT116, HeLa-S3, HepG2, IMR-90, K562	MCF-7	
CREB1	GM12878, H1-hESC, HepG2, K562	**MCF-7**	
CTCF	A549, H1-hESC, HeLa-S3, HepG2, IMR-90, K562, MCF-7	GM12878	**PC-3, induced_pluripotent_stem_cell**
E2F1	GM12878, HeLa-S3		**K562**
E2F6	A549, H1-hESC, HeLa-S3	K562	
EGR1	GM12878, H1-hESC, HCT116, MCF-7	K562	**liver**
EP300	GM12878, H1-hESC, HeLa-S3, HepG2, K562, SK-N-SH	MCF-7	
FOXA1	HepG2	**MCF-7**	**liver**
FOXA2	HepG2		**liver**
GABPA	GM12878, H1-hESC, HeLa-S3, HepG2, MCF-7, SK-N-SH	K562	**liver**
GATA3	A549, SK-N-SH	MCF-7	
HNF4A	HepG2		**liver**
JUND	HCT116, HeLa-S3, HepG2, K562, MCF-7, SK-N-SH	H1-hESC	**liver**
MAFK	GM12878, H1-hESC, HeLa-S3, HepG2, IMR-90	K562, MCF-7	
MAX	A549, GM12878, H1-hESC, HCT116, HeLa-S3, HepG2, K562, SK-N-SH	MCF-7	**liver**
MYC	A549, HeLa-S3, K562, MCF-7	HepG2	
NANOG	H1-hESC		**induced_pluripotent_stem_cell**
REST	H1-hESC, HeLa-S3, HepG2, MCF-7, Panc1, SK-N-SH	K562	**liver**
RFX5	GM12878, HeLa-S3, MCF-7, SK-N-SH	HepG2	
SPI1	GM12878	K562	
SRF	GM12878, H1-hESC, HCT116, HepG2, K562	MCF-7	
STAT3	HeLa-S3	GM12878	
TAF1	GM12878, H1-hESC, HeLa-S3, K562, SK-N-SH	HepG2	**liver**
TCF12	GM12878, H1-hESC, MCF-7, SK-N-SH	**K562**	
TCF7L2	HCT116, HeLa-S3, Panc1	MCF-7	
TEAD4	A549, H1-hESC, HCT116, HepG2, K562, SK-N-SH	MCF-7	
YY1	GM12878, H1-hESC, HCT116, HepG2, SK-N-SH	K562	
ZNF143	GM12878, H1-hESC, HeLa-S3, HepG2	K562	

Figure 14: Overview of transcription factors and cell types. As can be seen in the figure each row lists a transcription factor along with the cell types that can be used for training the model and the held out cell types. Source: Encode-Dream Transcription Factor Binding Challenge [4]

ChIP-Seq

The transcription factor binding data is provided as experimental data obtained from ChIP-seq experiments in two ways. First, the fold change over control, which is a signal over the genome, where a higher signal correlates with a transcription factor binding event. The transcription factor peak data provides the positions of the transcription factor binding events at 200bp resolution which is used to derive the labels for the prediction task. The processing pipeline to obtain the fold change and peak data from the alignments can be found in the following Github repository `https://github.com/kundajelab/chipseq_pipeline`. The peak data is provided in the narrow peak format [5]. The Irreproducible Discovery Rate (IDR) method [45] is used to score the reproducibility of peaks. Each peak computed by the peak caller is assigned a score representing the probability that the peak is noise. Peaks passing the 5% IDR threshold are defined as conservative peaks. Peaks that do not pass this threshold are marked as ambiguous peaks. All other regions in the genome are marked as unbound. The fold change data is provided in the BigWig format [37], which allows for compressing regions with no change in the signal.

DNA Sequence and Shape

The reference genome used is the human genome hg19 assembly [42], which is provided in Fasta format.
The geometries of base pairs and base steps (between consecutive bases) can be described by six variables, three translations and three rotations. These variables define the position and orientation of base pairs relative to the previous base pair along the axis of the helix and by that define the structure of the nucleic acid molecule. DNA shape features are calculated using the sequence data using the DNAShapeR package in R [18] which uses a sliding 5 base window to calculate four structural features from all-atom Monte Carlo simulations, displayed in Figure 15. If we recall the shape of the DNA, the strand backbones will be closer on one side than on the other side of the helix. The major groove width (MGW) is the with of the space where the backbones are far apart. DNA binding proteins can more easily access bases on the major grooves side since the backbone is not in the way. The roll and helix twist are both base step parameters, whereas the propeller twist and MGW are base pair parameters. In the prediction task, we pad the sequences with a non-determined base when calculating the base step parameters, since the first position will otherwise be undefined.

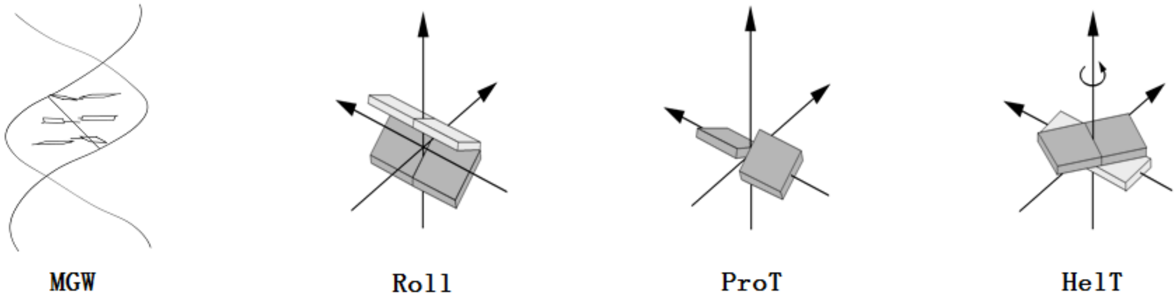


Figure 15: Structural DNA features computed by the DNAShapeR package. Source Zhou et al. University of Southern California [75].

Mathelier et al. show that adding these features improves in vivo binding site prediction [48].

DNase-Seq

TF binding events usually occur when the chromatin is accessible, that is the region does not have a nucleosome. Open chromatin can be experimentally captured by finding regions that are hypersensitive to DNase-I using DNase-Seq. The DNase-Seq data can thus be used as a cell type specific input for the model, allowing the model to generalize to new cell types. The protocol for processing the raw DNase data can be found here `https://github.com/kundajelab/atac_dnase_pipelines`. Provided are alignments in the BAM format [44], the fold change over control in BigWig format as well as the peaks in the narrow peak format.

RNA-Seq

The gene expression is estimated from RNA-Seq data using the ENCODE long RNA processing pipeline `https://www.encodeproject.org/rna-seq/long-rnas/`. For each cell type two biological replicates are provided with expression levels for 57820 genes as transcripts per million (TPM).

5.2 Prediction Task

The model estimates the probability of the transcription factor binding event occurring for a particular cell type for each 200bp region. The models are scored on four metrics.

- Area under receiver operator curve (AUROC)
- Area under the precision recall curve (AUPRC)
- Recall at false discovery rate of 50% (FDR50)
- Recall at false discovery rate of 10% (FDR10)

The receiver operator curve is generated by varying the threshold and calculating the false positive rate (FPR) and true positive rate (TPR).

$$\text{TPR} = \frac{tp}{tp + fn} \qquad\qquad \text{FPR} = \frac{fp}{fp + tn} \tag{41}$$

Similarly, the precision recall curve is calculated by varying the threshold and calculating the recall (true positive rate) and precision.

$$\text{precision} = \frac{tp}{tp + fp} \tag{42}$$

The false discovery rate is defined as the complement of the precision.

The performance of the methods in the competition is validated in three ways. In the leaderboard round the models are scored on held out cell types on the chromosomes 1, 8 and 21, which amounts to approximately 8 million bins per cell type. In the final round the models are scored on the full genome of a held out cell type totaling approximately 60 million bins. Lastly, in the benchmark round the performance of the models is measured on the missing chromosomes 1, 8 and 21 of training cell types.

For our internal validation, we held out one cell type for testing and used the other cell types for training.

5.3 Training and Validation

The internal validation phase was done mostly using transcription factor CTCF using the validation pipeline described above. For each cell type, the training data consists of more than 50 million training instances. Most of these training instances will not add any information to the model, since a lot of non-binding and non-open regions will be redundant. In fact of the ~ 50 million train regions, only around 500k will have binding events. To save training time, we choose to train on regions that overlap with DNase peaks augmented with 100k regions per cell type that are not hypersensitive to DNase-I. For CTCF this results in ~ 2.5 million train regions for all the train cell types combined.

We found that the training in most cases quickly started stagnating or overfitting the train data. One possible improvement we tried was to initialize the filters with known transcription factor sequence motifs from the JASPAR database, however this did not improve performance. We observed small performance improvements in AUC by pre-training the network using the mean fold change of the ChIP-seq signal as the target for prediction, in essence converting it to a regression task, although we did not use pre-training for our submitted models.

5.4 Feature Engineering

From the data provided, the sequence, DNase fold change, the four shape features and gene expression were used in experiments.

The sequence is converted to a one hot encoding as described in the background section where regions that are not sequenced are replaced by zeros. The fold change signal from the BigWig files were extracted using WiggleTools [72]. The shape data was computed using DNAShaper and lastly we used the TPM values from gene expression data for the 32 TFs of the challenge. Both the shape and gene expression data was normalized using zero mean and unit variance. The open chromatin data was transformed by applying the function $x \rightarrow \log(x+1)$ to each data point.

The sequence encoding, shape features, open chromatin signal over the genome and labels were saved into binary formats such that they can be loaded into memory fully which allows for fast batch generation. Through experimentation we found that mixing training and cell types during training improves the convergence of the model. Hence each batch is constructed by taking train regions from different regions of the genome and different cell types. Using Keras' batch generator, batches with training examples can be constructed on the CPU while the network is trained on the current batch using the GPU. The disadvantage of using this approach is that the entire genome, accessibility, shape and label data needs to be loaded into RAM as using a memory mapped file is prohibitively slow in generating train batches as we want the GPU to be utilized at all times. Loading all data into memory requires at least 128GB of RAM.

In order to make a prediction for a particular 200bp bin, we include the regions 400bp upstream and downstream of the bin as enhancers are 50-1500bp in length.

5.5 Network Architecture

The architecture used for submissions is displayed in Figure 16. The architecture is quite shallow having one convolutional layer where large filters are used. We use separate paths for the different features and merge them in a higher level, where the combined features are processed using a fully connected layer. Similar to the sequence processing path, the shape can also be processed using convolutional/pooling layers. The gene expression data can be processed using a dense layer.

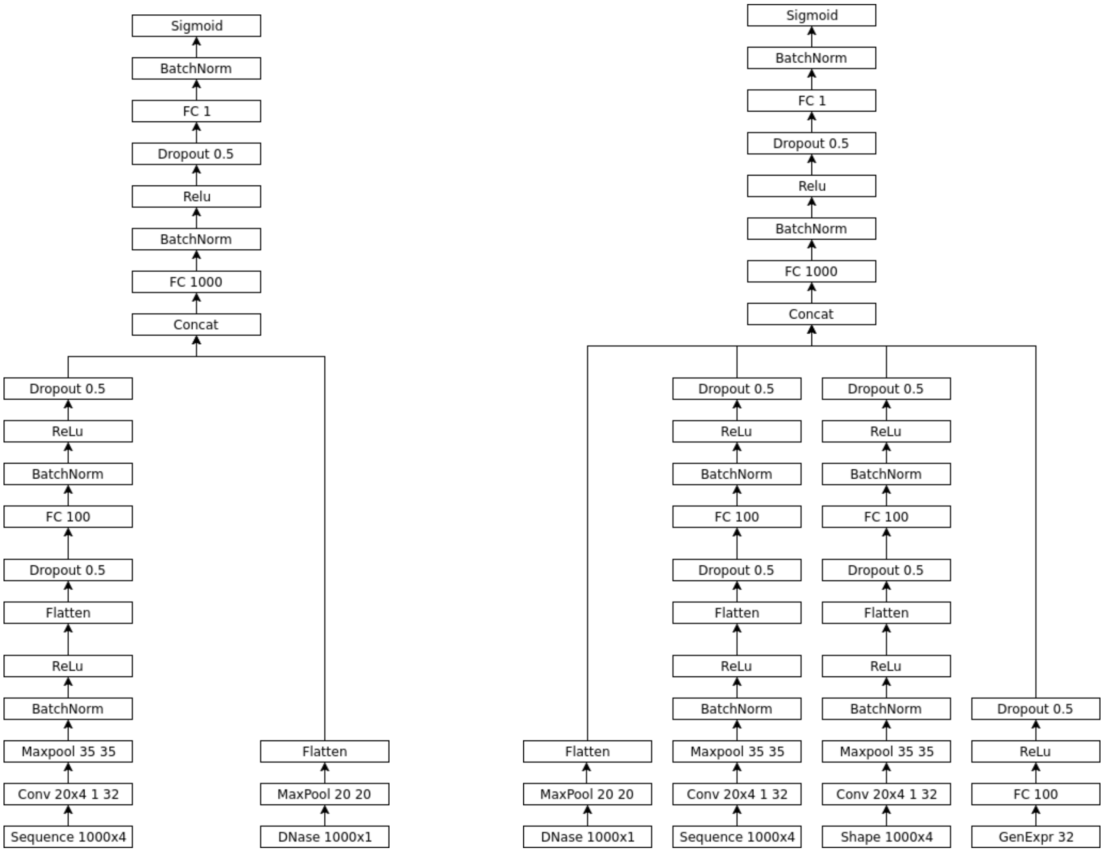


Figure 16: Left: architecture using for making submissions to challenge. Right: Architecture can be extended with new data.

In images, it was found that a deeper architecture with convolutional filters with smaller receptive fields results in improved performance [68]. Figure 17 shows the deeper architecture we used in our experiments.

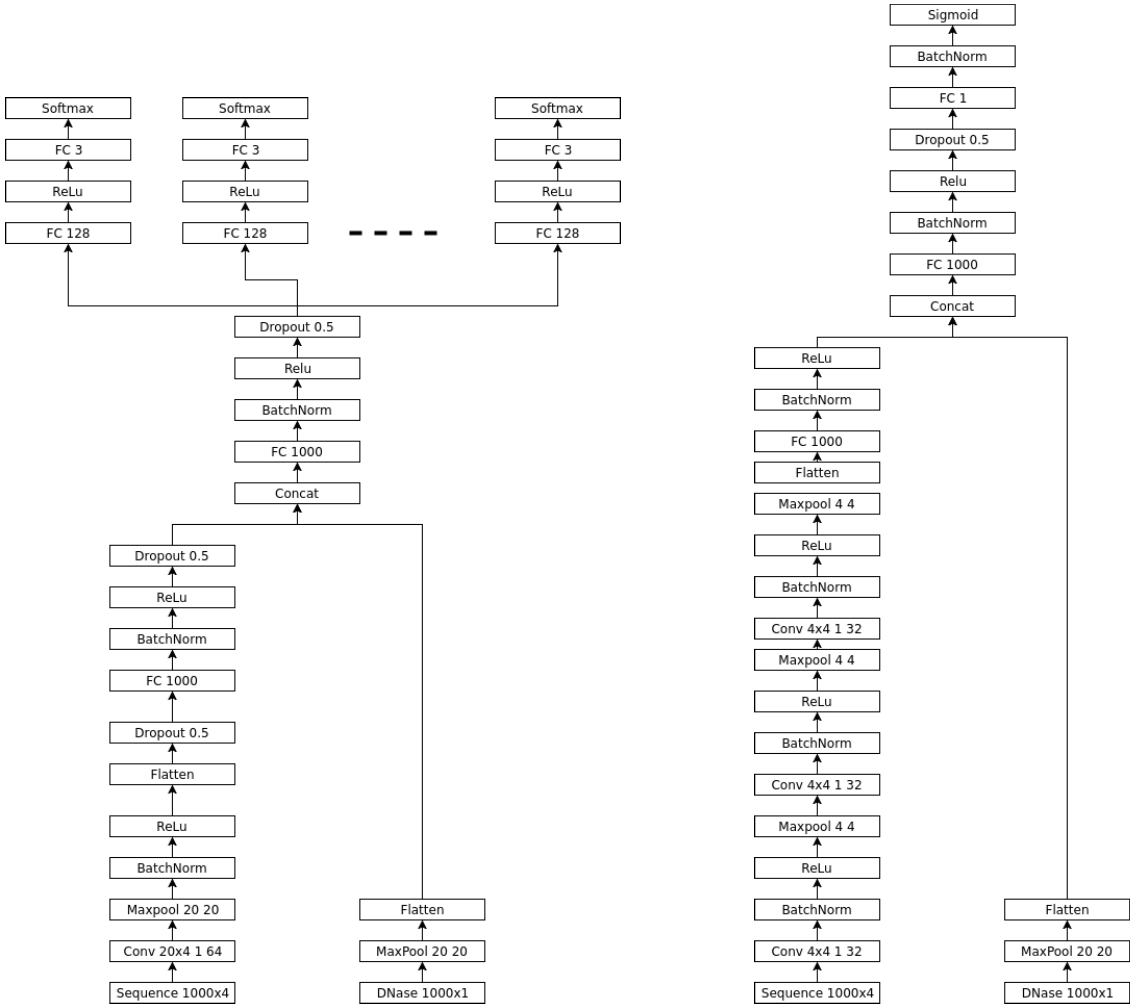


Figure 17: Left: Multi-Task Network. Right: Deep architecture.

Instead of building one model for each transcription factor, predicting the binding sites for the transcription factors can be combined. Figure 17 show the joint model we used to make predictions for all transcription factors simultaneously. In particular, for each task (TF) there are three nodes present. The first node represents that the fact that the region is unknown to be binding site. The second node represents non-binding and lastly the third node represents binding. The error is then calculated using a weighted cross entropy loss function, where the weight of unknown sites are set to 0, non-binding is set to 1 and binding is set to the ratio of non-binding to binding. By doing this, the error propagates solely from TFs for which we have data for that particular combination of genomic region and cell type, which means that only TFs for which there is data will impact the representation. The idea of using this architecture is that a general shared representation for sequence and DNase is created, which is then transformed for each TF individually. That is, each transcription factor shares a representation of the DNase and sequence of 1000 latent factors and computes a representation specific to that transcription factor using 128 hidden units which are not shared between the transcription factors. The representation learned for a particular TF can thus be leveraged to improve the representation for other TFs.

For optimization of the network parameters we used the ADAM algorithm using a learn rate of 0.001 and a batch size of 256. The single-task models (the models for an individual TF) are trained for a maximum of 50 epochs with a patience of 10. The multi-task model was trained for 5 epochs. The network parameters were initialized using Xavier initialization [27].

5.6 Implementation

The convolutional neural networks were implemented in Python using the Keras package with TensorFlow. The code is available on `https://github.com/nielstendijke/Dream_TF_net` and includes code to programmatically download and pre-process the data as well as train and models and submit predictions. The hardware used for the experiments consists of an eight core Xeon workstation with 128 GB RAM and four GPUs (1x GTX Titan X 12GB, 3x 1070 8GB).

5.7 Results

This section summarizes the results obtained on the internal validation set (Table1 4), the leaderboard round (Table 6), final round results (Table 7) and the benchmark round (Table 8). In the leaderboard, final and benchmark round we used the shallow architecture and the sequence + DNase features. The naive method is a linear classifier which classifies all regions which are not open as non binding and uses scores obtained from known TF binding motifs and the max DNase fold change on open chromatin regions. Autosome is the winning method on the first round of the challenge and Nabla is our submission. As can be seen in the results, our method performs well on CTCF (on par or better than Autosome), but performs relatively poorly on other transcription factors. As expected the benchmark round produces the highest AUC and AUPRC values on average, since these predictions are made within cell type, that is the test data are the held out chromosomes of the training set for that particular cell type. Our method performs poorly on the final benchmark, most of which is evaluated on the liver cell type.
Figure 3 shows the training progress for each of the configurations of Table 4. Most of the progress is made is the first 10 epochs after which it starts stagnating or even overfitting on the validation set. Note that the validation set here refers to a part of the training set, which is not used to train the model but is solely used to decide when to terminate the training procedure.

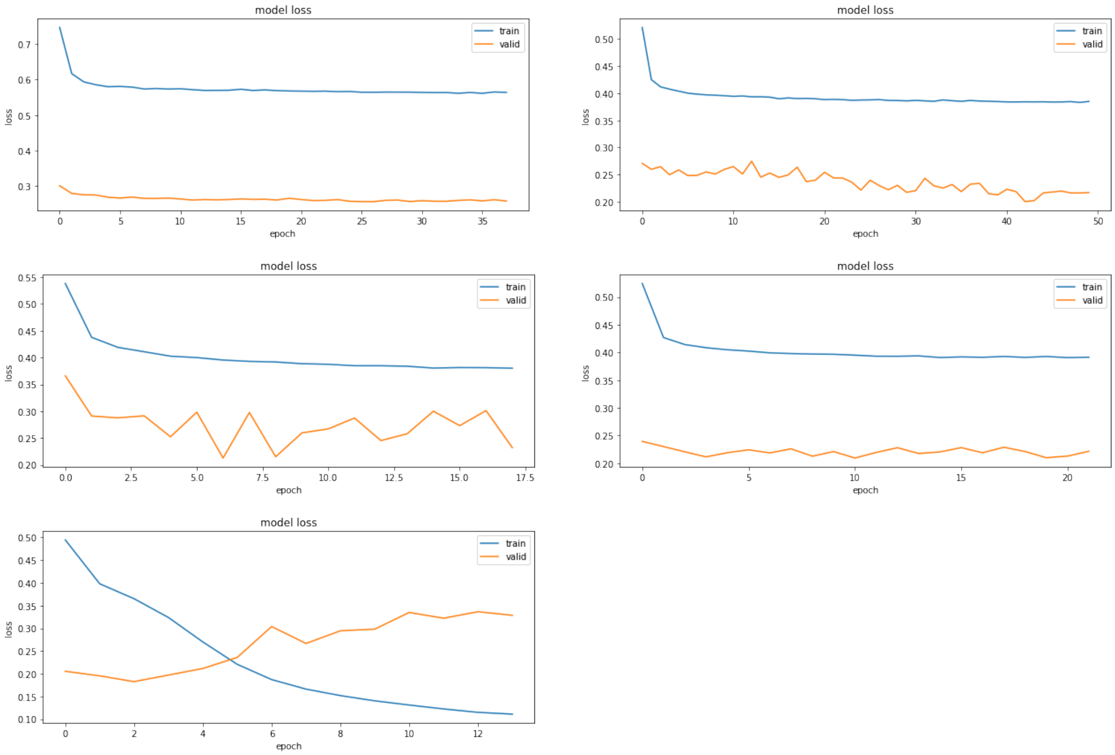


Table 3: Training progress for each of the configurations of the single task models. From left-to-right, from top-to-bottom: Shallow (sequence), Shallow (sequence + DNase), Shallow (sequence + DNase + shape), Shallow (sequence + DNase + gene expression), Deep (sequence + DNase). The train set is represented by the blue line and the validation set is represented by the orange line.

Model	Features	AUC	AUPRC	FDR50	FDR10	Time/epoch (s)
Shallow						
	Sequence	0.950	0.263	0.200	0.012	410
	Sequence + DNase	**0.988**	0.655	0.700	0.343	450
	Sequence + DNase + Shape	0.987	**0.662**	**0.701**	**0.360**	780
	Sequence + DNase + Gene expr	0.985	0.633	0.673	0.307	460
Deep						
	Sequence + DNase	0.919	0.459	0.469	0.127	560

Table 4: Results obtaining using the validation pipeline on the transcription factor CTCF. The scores are averaged over two different held out cell type sets.

Cell type	AUC	AUPRC	FDR50	FDR10
JUND	0.969	0.163	0.006	0.0
EGR1	0.979	0.141	0.001	0.0
MYC	0.99	0.29	0.098	0.0
GABPA	0.924	0.147	0.0	0.0
REST	0.924	0.031	0.0	0.0
RFX5	0.928	0.03	0.0	0.0
TCF12	0.943	0.076	0.0	0.0
CTCF	0.963	0.382	0.324	0.04
ATF2	0.644	0.006	0.0	0.0

Table 5: Results obtaining using the validation pipeline using the multi-task network. One celltype is held out and the others are used for training.

Cell type	Method	AUC	AUPRC	FDR10	FDR50
ATF7	Naive	0.6833	0.0364	0.0002	0.0002
	Nabla	**0.9357**	0.1736	0.0091	0.1035
	Autosome	0.8971	**0.3256**	**0.0772**	**0.2988**
CREB1	Naive	0.7426	0.1803	0.0016	0.1120
	Nabla	**0.8569**	0.2335	0.0200	0.1886
	Autosome	0.8145	**0.2656**	**0.0476**	**0.2324**
CTCF	Naive	0.7357	0.2237	0.0073	0.1983
	Nabla	**0.9892**	0.6965	0.4601	0.7109
	Autosome	0.9879	**0.7327**	**0.5292**	**0.7380**
EP300	Naive	0.9391	0.0977	0.0000	0.0000
	Nabla	0.9697	0.1331	0.0000	0.0007
	Autosome	**0.9891**	**0.2702**	**0.0003**	**0.1348**
JUND	Naive	0.9115	0.1754	0.0003	0.0221
	Nabla	0.9850	0.2465	0.0087	0.1119
	Autosome	**0.9938**	**0.4040**	**0.0406**	**0.3066**
TAF1	Naive	0.8764	0.3446	0.0000	0.0000
	Nabla	**0.9828**	0.5499	0.0001	0.6270
	Autosome	0.9796	**0.5861**	**0.0822**	**0.6560**

Table 6: Leaderboard round results.

Cell type	Method	AUC	AUPRC	FDR50	FDR10
CTCF	Nabla	0.9206	**0.5024**	**0.4881**	0.3004
	Autosome	**0.9371**	0.4800	0.4369	**0.3062**
E2F1	Nabla	0.9695	0.2593	0.004793	0.0
	Autosome	**0.9852**	**0.4511**	**0.4623**	**0.0318**
EGR1	Nabla	0.9518	0.1755	0.0795	0.000
	Autosome	**0.9814**	**0.3581**	**0.2276**	**0.0172**
FOXA1	Nabla	0.8900	0.1865	0.1443	0.0019
	Autosome	**0.9374**	**0.3046**	**0.2657**	**0.027**
FOXA2	Nabla	0.9276	0.1737	0.0972	0.0004
	Autosome	**0.9559**	**0.348**	**0.3155**	**0.0037**
GABPA	Nabla	**0.9447**	0.2809	0.256	0.0255
	Autosome	0.9364	**0.4372**	**0.3608**	**0.0938**
HNF4A	Nabla	0.8877	0.301	0.2576	0.0275
	Autosome	**0.903**	**0.5191**	**0.5479**	**0.1573**
JUND	Nabla	0.9303	0.1804	0.0673	0.004
	Autosome	**0.9671**	**0.4853**	**0.5185**	**0.0509**
MAX	Nabla	0.9524	0.2221	0.0884	0.0001
	Autosome	**0.9724**	**0.4615**	**0.5007**	**0.0028**
NANOG	Nabla	0.9335	0.0616	0.0	0.0
	Autosome	**0.9577**	**0.2438**	**0.181**	**0.0025**
REST	Nabla	0.9114	0.2251	0.1525	0.0023
	Autosome	**0.9144**	**0.3204**	**0.2613**	**0.0159**
TAF1	Nabla	0.9472	0.219	0.1021	0.0
	Autosome	**0.9776**	**0.3619**	**0.2338**	**0.0007**

Table 7: Final round results.

Cell type	Method	AUC	AUPRC	FDR50	FDR10
CTCF	Nabla	**0.9712**	**0.6235**	**0.6405**	**0.3163**
	Autosome	0.9529	0.5561	0.5361	0.3282
E2F1	Nabla	0.9699	0.4246	0.4613	0.0093
	Autosome	**0.9941**	**0.5161**	**0.5476**	**0.0563**
EGR1	Nabla	0.976	0.3945	0.3749	0.0039
	Autosome	**0.9934**	**0.5815**	**0.6379**	**0.0915**
FOXA1	Nabla	0.9308	0.2819	0.2501	0.0063
	Autosome	**0.9712**	**0.4173**	**0.4014**	**0.0544**
FOXA2	Nabla	0.941	0.3454	0.3293	0.0192
	Autosome	**0.9748**	**0.5064**	**0.5233**	**0.1378**
GABPA	Nabla	0.9691	0.4755	0.4878	0.0658
	Autosome	**0.9889**	**0.6202**	**0.6771**	**0.1964**
HNF4A	Nabla	0.9256	0.4652	0.4727	0.1541
	Autosome	**0.965**	**0.6293**	**0.6654**	**0.3111**
JUND	Nabla	0.9437	0.4251	0.444	0.0372
	Autosome	**0.9809**	**0.5968**	**0.664**	**0.1412**
MAX	Nabla	0.9574	0.4707	0.4932	0.085
	Autosome	**0.9868**	**0.6372**	**0.7039**	**0.204**
NANOG	Nabla	0.9675	0.2028	0.1013	0.0096
	Autosome	**0.9897**	**0.3591**	**0.3147**	**0.0443**
REST	Nabla	0.9352	0.4024	0.4209	0.0003
	Autosome	**0.9719**	**0.5865**	**0.6418**	**0.1618**
TAF1	Nabla	0.9602	0.3212	0.2828	0.0005
	Autosome	**0.9911**	**0.4644**	**0.4669**	**0.0061**

Table 8: Benchmark round results.

5.8 Discussion

In our own internal validation we tried (combinations) of several different features and configurations of the network. Using the DNase fold change signal has the largest positive effect over only using the sequence, which is expected since binding will mostly occur at DNase sensitive regions. Furthermore, the cost of adding the DNase features in terms of training time is relatively small. Note that obtaining the open chromatin data requires experimentation, though only once per cell type as opposed to once for each cell type and transcription factor combination. Adding the sequence features does not significantly improve performance and adds significant costs to both memory requirements (20GB) and training time (73% increase). The features derived from the RNAseq experiments also seem to add no useful information.

It is important to note that most experimentation was done on CTCF, which is known to have a sequence motif which is shared in most in vivo binding sites and is highly conserved in vertebrates. Furthermore, CTCF localization is largely invariant across cell types [38]. Unsurprisingly, our method performed best on CTCF, with only a few transcription factors on the leaderboard round performing on par with the winning method. In the final benchmark this disparity is even more clear. Note that the AUC and the AUPRC performance metrics tend to balance each other out. For example, at the expense of AUC, the AUPRC can be increased to 0.5 by setting all binding sites to be binding. The recall at the different precision thresholds correlates strongly to the AUPRC.

A major factor in the performance of the Autosome team was due to choosing which cell types to use to train. This was done by comparing the cell types on DNase accessibility and gene expression [3]. This may also explain the poor performance of the multi-task network, which is trained with all the cell types simultaneously. In particular the cell type specific features, that is the features to extrapolate to new cell types are the DNase accessibility and the gene expression data. Since we did not add any further identifying information, it may not be easy for the CNN to learn the relationships between cell types. A simple fix for this is to add a one hot feature vector identifying the cell type, though not sufficient since some cell types will not have training data available.

As for CNN architecture, we found shallow models with large filters to result in the best performance. This is very different from image models such as the standard CIFAR-10 model, where having multiple convolutional layers improves performance. A major advantage of using the CNN models is that sequence and DNase features are found automatically, whereas the Autosome team manually created features from the sequence, DNase and RNAseq data and fed those features to a XGBoost model [17].

For future work, more time should be spent on the multi-task network since it can leverage the entire training set of transcription factors to build better representations. Additionally, it is computationally much more efficient to build one model which can train and make predictions on multiple transcription factors simultaneously than to build a model for each transcription factor separately. Adding the Autosome improvement of choosing which training cell types to use could be accomplished by having separate representations for each subset of cell types. Additionally, more experimentation could be done on selecting the regions of the DNA which are used for training. Lastly, more experimentation could be done on the architecture of the models, e.g. adding recurrent layers. Ideally, this should be mostly done on the multi-task network in order to avoid having to tune a model for each individual transcription factor.

6 Experiment: Transcription Start Site Prediction

The transcription start site (TSS) is the 5' end of the genome where transcription from DNA to messenger-RNA starts. As mentioned in Section 2, the TSS sites can be experimentally determined at base pair resolution using CAGE technology.
The FANTOM5 project [22] contains 573 primary cell samples for which CAGE tags were collected. The tags were then clustered based on proximity, where clusters larger than 49 bp were separated into non-overlapping regions based on expression profiles using independent component analysis [32]. This resulted in ≈3.5 million peak regions. These peaks are further filtered down to a 'robust' set of peaks which were well supported by at least 11 observations resulting in ≈200 thousand peaks. In the literature, a distinction is made between sharp peaks and broad peaks [22]. Sharp peaks have a short peak region where the tags are close together and have TATA box enrichment. The CAGE tags corresponding to broad peaks are more dispersed and these peaks will have a larger peak region and are CG-enriched. In addition, sharp or focused peaks are more correlated with tissue-specific expression, whereas broad peaks are correlated to broad expression throughout the organimsal cycle.

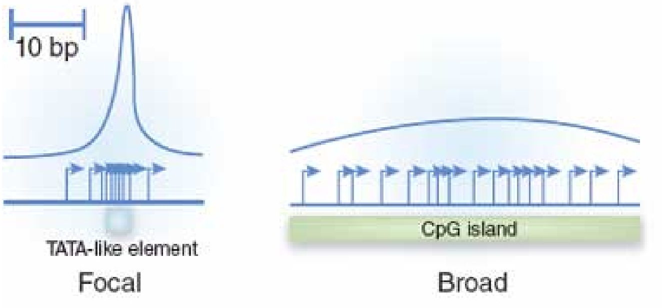


Figure 18: Sharp CAGE peak (left), broad CAGE peak (right). Source: Illuminating eukaryotic transcription start sites, Stamatoyannopoulos [64].

In this experiment, we built a predictive model for the CAGE peaks using a convolutional neural network and interpreted this trained model. For the prediction task, we define a CAGE peak to be a sharp peak if its peak region is of length 11 bp or shorter and a broad peak otherwise.

6.1 Approach

Convolutional neural networks can be used to build predictive models of the experimentally determined TSSs. In the experiment, primary sequence is used to build a predictive model of the CAGE peaks. More specifically, for a specific genomic region, the model predicts whether the exact center of that region corresponds to a sharp peak, broad peak or the background.
The convolutional neural network used is depicted in Table 9. This architecture was found by starting from a shallow network, i.e. having one convolution/pool layer and increasing the number of convolutional layers until performance did not improve. For this task, having more than one convolution/pool layer already does not improve performance. The parameters of the network were trained using the Adam optimization algorithm with a learning rate of 0.001 and a batch size of 256 using the cross entropy error loss.
We created a dataset by augmenting the CAGE peak locations with non-peak (background) locations chosen uniformly at random from the non-peak regions of the genome. This resulted

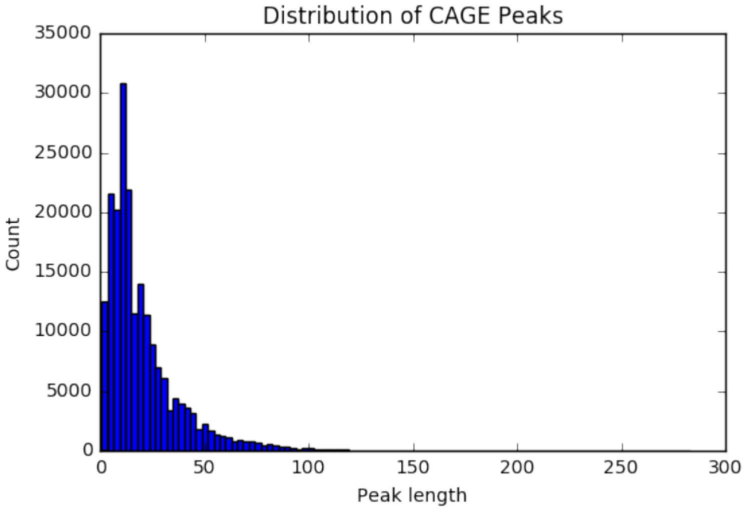


Figure 19: Distribution of CAGE peak lengths in the dataset.

in a dataset where 10% (201,802) of the data points are peaks and 90% (1,816,218) are non peaks. Note that the peaks can be on either strand. In the case the peaks are on the negative strand, the reverse complement for that region is taken as input. That is, the sequence is flipped and each base is replaced by its complementary base, i.e. A is replaced by T and C is replaced by G and vice versa. The dataset was then split into a train and held-out validation set by holding out chromosomes 1, 8 and 21 and using the rest for training.
Before training the train set is first partitioned into a train set (80%) and validation set (20%). The network is trained for a maximum of 100 epochs with an early stopping patience of 10 using the validation loss as the benchmark.

Layer	**Description**
Input	200x4 DNA sequence
1D Convolution, width 10	Stride 1, valid padding, output 191x16
1D Max Pooling, width 4	Stride 4, output 47x16
ReLU	
Flatten	output 752
Fully connected	output 128
ReLU	
Dropout	rate 0.5
Fully connected	output 3
Softmax	Propabilities for each class, output 3

Table 9: Architecture of the CNN used for TSS prediction.

Each nucleotide of the DNA sequence is one-hot encoded. Next the one-hot encoded sequence is processed by a convolutional layer, which has 16 filters of length 10 and uses a stride of one. The activations of the convolutional layer are then downsampled using max-pooling. The downsampled activations are then passed through a ReLU non-linearity. This is then followed by a dense layer with 128 units and a dropout layer with a dropout rate of 0.5. Lastly, there are 3 output units corresponding to the three output classes: sharp peaks, broad peaks and the background.

Next, activation maximization and the DeepLIFT method are both applied to the trained network. The starting point for activation maximization is chosen to be the matrix with 0.025 for all the entries. L2-regularized gradient ascent with a learning rate of 0.01 and λ of 0.01 using 1000 iterations is then applied to find an input which maximizes a particular class, i.e. sharp or broad peak. Note that the activation before applying softmax is maximized, since the softmax output can be maximized by minimizing the probability for the other classes. For the DeepLIFT method, the reference is chosen to be the background frequencies of the nucleotides in the region of 2000bp upstream and 200 bp downstream of the TSS [67].

Lastly, we compare the results with the current default tool to find motifs, MEME-ChIP. MEME-ChIP is run in discriminative mode, with the the different peaks (broad and sharp) as the primary sequences and non-tss sites as the control sequences.

6.2 Results

As can be seen in the figure below, the validation stops decreasing after about 10 epochs. The train error is still decreasing, which means the model is overfitting.

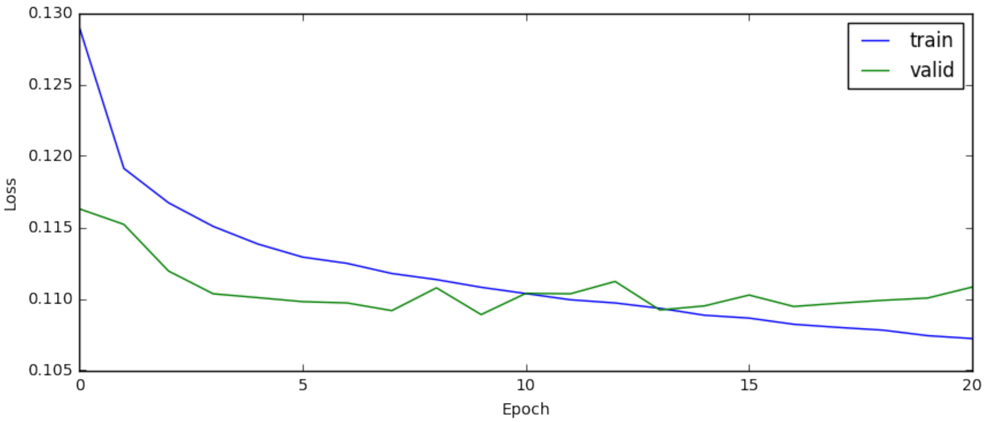


Figure 20: Train progress.

The confusion matrix in Figure 21 shows that the network easily distinguishes peaks from non-peaks, however, many sharp peaks are classified as broad peaks.

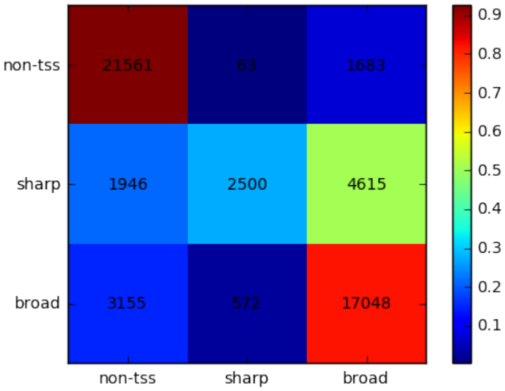


Figure 21: Confusion matrix of the predictions.

Convolutional Filters

The convolutional filters are visualized as sequence logos in Figure 22. In order to make the filters better interpretable, the negative parts of the filters were zeroed out. Note that this is precisely the result obtained when applying the activation maximization procedure, where the individual filter responses are maximized. It can be seen that multiple TATA box recognizing filters are learned, showing that some redundancy in the filters and that the number of filters could be decreased.

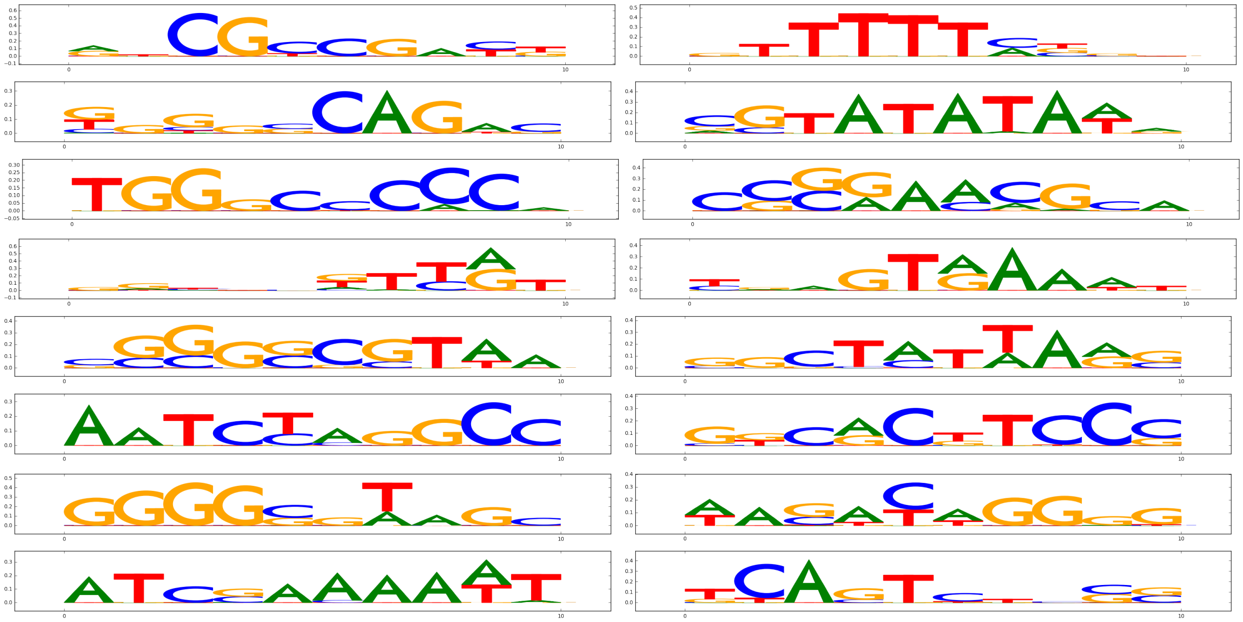


Figure 22: Convolutional filters visualized as sequence logos.

Activation Maximization

Figure 23 shows the inputs that maximize the sharp and broad peak classes. The network seems to pickup on the TATA box, which a well-known cis-acting regulatory element occurring in about 20% of human promoters. The increased CG content for the broad peak is also expected and reported in literature. It must be stated though, that although the TATA box seems to be a recurring pattern when using activation maximization, there will be a lot of variance within multiple runs depending on the initialization of the parameters of the network.

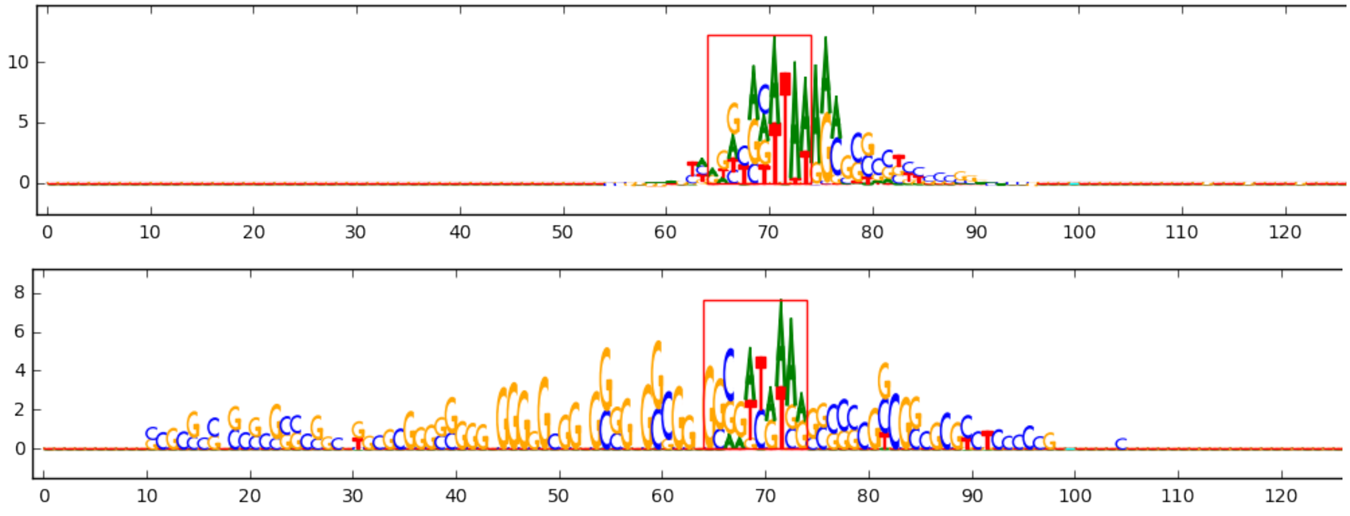


Figure 23: Max class models of sharp TSS peaks (top) and broad TSS peaks (bottom). The transcription start site is at the exact center (ending at location 100). The area highlighted in red is where the TATA box is expected. Scores below the 85-th percentile are set to zero in order to get a more clear picture.

DeepLIFT

We calculated the contributions of the inputs on the distance to reference for each example in the training data set. The mean of the contributions for all the examples is depicted in Figure 24. As expected, the TATA box region contributes more to the score for the sharp peaks, as the TATA box will be enriched for sharp TSSs. Another interesting observation is that the contributions in the center seem to be more important than the TATA box region, which to the best of our knowledge has not yet been reported in the literature.

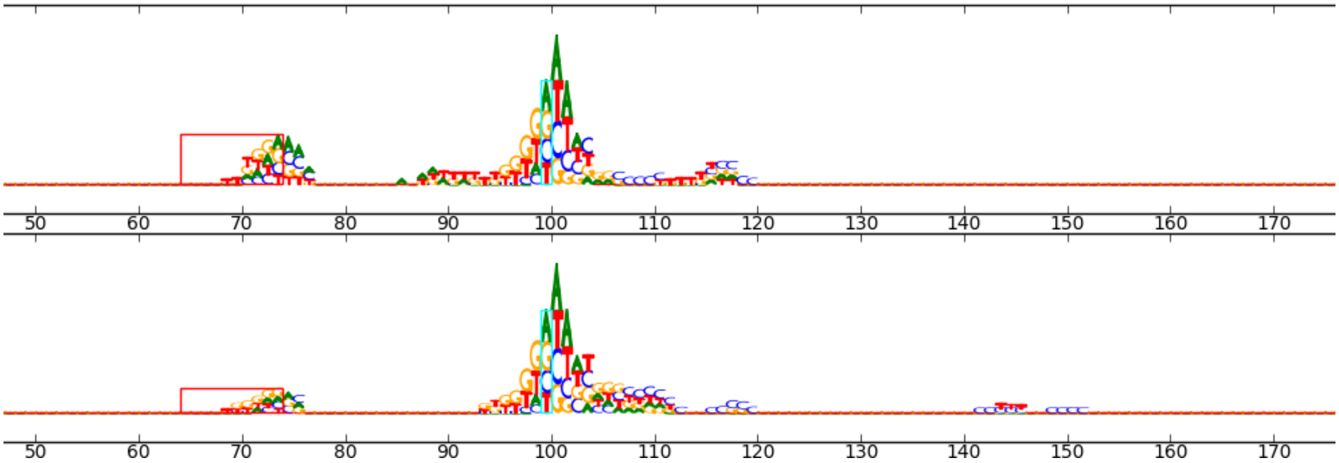


Figure 24: DeepLIFT contributions for sharp peaks (top) and broad peaks (bottom). The area indicated by the red rectangle is the location where the TATA box is expected, which has higher contributions in the sharp peak class compared to the broad peak, consistent with the literature. The center however, seems to contribute much more to the network classifying it as a TSS peak.

We tested this hypothesis by performing in silico mutagenesis, where a specific part of the sequence is left blank, i.e. replaced by zeros during traning and the resulting AUC and AUPRC on the test set was observed. Note that since there are three classes the AUC and AUPRC are calculated using a one-vs-all approach. The scores can be found in Table 10.

Safe region	Non-TSS		Sharp		Broad	
	AUC	AUPRC	AUC	AUPRC	AUC	AUPRC
Entire input	0.954	0.944	0.798	0.545	0.889	0.806
TATA box	0.813	0.787	0.680	0.362	0.769	0.671
Center	0.909	0.898	0.750	0.479	0.844	0.740
TATA box + center	0.937	0.925	0.788	0.540	0.870	0.780

Table 10: Results of the in-silico mutagenesis. The 'Safe region' is the region in the input that is *not* zeroed out. The TATA-box region is defined as 35bp to 20bp upstream of the TSS. The center region is defined as 10bp upstream to 10bp downstream of the TSS. The TATA box + center region is defined as the region 35bp upstream to 10 bp downstream of the TSS.

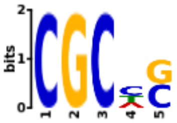
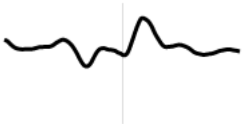
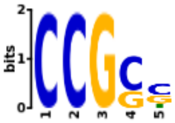
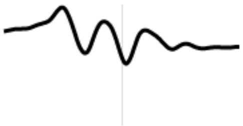
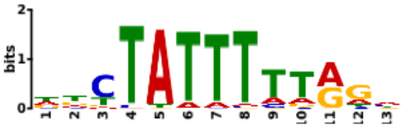
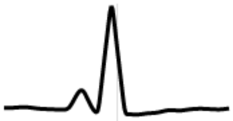
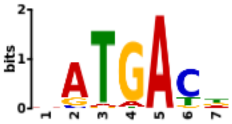
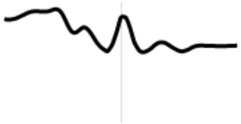
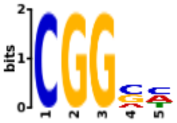
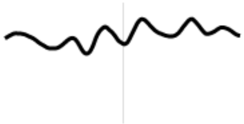
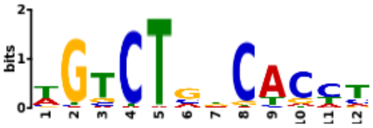
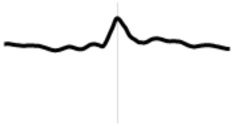
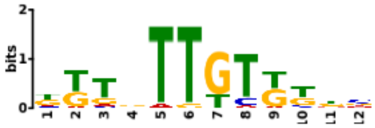
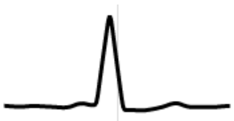
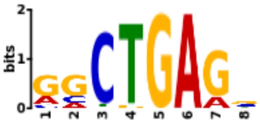
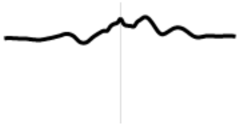
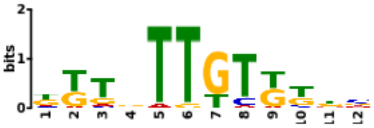
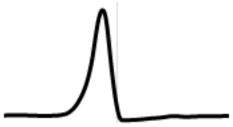
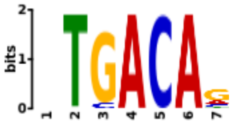
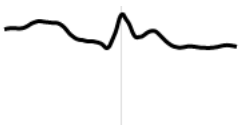
Motif	Distribution	Motif	Distribution
			
			
			
			
			

Table 11: Top five sharp motifs (left) and broad motifs (right).

MEME-ChIP

Figure 11 shows the top five motifs found by MEME-ChIP along with their distribution profiles along the 200bp region. As can be seen, the sharp TSS motifs are more focused than the broad motifs.

6.3 Discussion

By visualizing the filters the motifs found by the network can be visualized as sequence logos. Unlike the motifs found by MEME-ChIP, the filter sizes are fixed. However, larger motifs can be learned by combining filters.

The distributions of the motifs in the MEME-ChIP results seem to indicate that the sharp TSS motifs are more focused around the center, whereas the broad TSS are more dispersed. Looking at the DeepLift results, we see a similar picture. The contributions of the center and TATA box are larger for the sharp class compared to the broad class. In particular, the center region seems to have the largest impact on the center. Note that the DeepLift results are aggregated over all examples, which means that positive and negative contributions could cancel each other out. Therefore, we validated this result with the in-silico mutagenesis, where a higher AUC and AUPRC were observed when keeping the center fixed as compared to keeping the TATA box region fixed. However, unlike the TATA box motif, which is found by visualizing the motifs, activation maximization and MEME-ChIP, the center does not have a clear motif.

We found that a shallow architecture performs best on this task in line with other work applying convolutional neural networks to predict chromatin effects from sequence only [71]. DeepChrome [60], which uses histone modification data to predict TSS sites uses a similar architecture, which was found using a grid search. Seemingly, the peaks as identified by the CAGE data can be described by linear combinations of the motif detectors, although some experimentation with recurrent bidirectional LSTM layers did provide small improvements to the AUC and AUPRC scores. As future work, we could investigate the cases where LSTMs perform better as well as increase the length of the upstream region as enhancer regions may not be captured in this short window. The current version of DeepLift does not support recurrent layers, however this should be a straightforward task to implement using the the rules for affine functions and non-linearities.

As the CAGE peaks in this study are aggregated for all cell types a future extension on this experiment could be to study cross-cell type prediction using cell-type specific data such as histone modifications and open chromatin.

7 Summary and Conclusion

Following the completion of the Human Genome Project in 2003, next-generation sequencing allow for rapid sequencing of DNA and RNA and by that have revolutionized genomics and molecular biology. Many regions of the DNA which do not code for proteins contain regulatory elements such as promoters, enhancers, insulators, etc, which play crucial roles in gene expression. Precisely identifying and characterizing these regions could improve our understanding of the transcription process and lead to new medicines for genetic diseases. NGS technology allow for identifying and studying the genomic factors that are involved in these events such as transcription factor binding, histone modifications and open chromatin with great sequencing depth. Furthermore, these data allow researchers to build predictive models for these events using machine learning approaches, which allow for annotating new cell types without having to perform the experiment.

Recently, deep convolutional neural networks have proven to be very successful on many artificial intelligence tasks such as image classification, finding policy and value functions for game playing AI and drug discovery. CNNs are characterized by having spatially local connections. This connectivity pattern allows CNNs to be effective on data that have a grid-like topologies such as images and DNA sequences.

In this work we conducted two experiments. In the first experiment, transcription factor binding site prediction was studied by evaluating different features generated from sequence, shape, open chromatin and gene expression data and various CNN architectures to model these data. We found that sequence and open chromatin are the most important factors to consider when building models for transcription factor binding. As for architectures, we found that shallow models with large filter sizes resulted in the best predictive performance. Building a model for each individual transcription factor outperformed our multi-task approach, where we built a CNN with a shared representation of the sequence and open chromatin for all the transcription factors. In comparison to other methods, we found that our CNNs performed close to the state of the art on some transcription factors and cell types, while being significantly worse on others. This is mainly due to the selection of cell types for training and train regions, rather than the method itself.

In the second experiment, we built CNN models to exactly predict the location of the transcription start site from CAGE data. The trained models were then interpreted, which lead to the finding that the area directly around the TSS site is most decisive factor for determining whether a particular base is a TSS, which to best of our knowledge is not reported in literature.

In conclusion, we found that NGS data can be used to build predictive models of in-vivo transcription factor binding sites which can extrapolate to new cell types and thereby complement experimental findings. Convolutional neural networks are a flexible way of modeling genomic data without the need for much pre-processing or manual feature creation. As future work, more study could be done on the architecture of the neural network. In particular, multi-task networks, deeper networks and recurrent layers.

References

[1] Cage Tags, Kabushiki Kaisha DNAFORM. https://cage-seq.com/, 2017. [Online; accessed 03-April-2017].

[2] Chromatin structure, The O'Sullivan Lab, UPCI, Pittsburgh. http://osullivanlab.squarespace.com/about-chromatin/, 2017. [Online; accessed 03-April-2017].

[3] Dream challenge write-up by the Autosome team of the Engelhardt Institute of Molecular Biology. http://autosome.ru/slides/DREAM.ENCODE.autosomeru.method.writeup.pdf, 2017. [Online; accessed 28-May-2017].

[4] Encode-Dream Transcription Factor Binding Challenge. https://www.synapse.org/#!Synapse:syn6131484/wiki/402033, 2017. [Online; accessed 19-January-2016].

[5] Narrow Peak format. https://genome.ucsc.edu/FAQ/FAQformat.html#format12, 2017. [Online; accessed 03-April-2017].

[6] OxfordNet Architecture. Fossard, University of Toronto. https://www.cs.toronto.edu/~frossard/post/vgg16/, 2017. [Online; accessed 03-April-2017].

[7] Schematic overview of transcription, Memorial University of Newfoundland. http://www.mun.ca/biology/desmid/brian/BIOL2060/BIOL2060-21/CB21.html, 2017. [Online; accessed 03-April-2017].

[8] Synapse. https://www.synapse.org/, 2017. [Online; accessed 17-June-2017].

[9] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[10] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831–838, 2015.

[11] Timothy L Bailey, Mikael Boden, Fabian A Buske, Martin Frith, Charles E Grant, Luca Clementi, Jingyuan Ren, Wilfred W Li, and William S Noble. Meme suite: tools for motif discovery and searching. *Nucleic acids research*, page gkp335, 2009.

[12] Jon-Matthew Belton, Rachel Patton McCord, Johan Harmen Gibcus, Natalia Naumova, Ye Zhan, and Job Dekker. Hi–c: a comprehensive technique to capture the conformation of genomes. *Methods*, 58(3):268–276, 2012.

[13] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[14] Bradley E Bernstein, John A Stamatoyannopoulos, Joseph F Costello, Bing Ren, Aleksandar Milosavljevic, Alexander Meissner, Manolis Kellis, Marco A Marra, Arthur L Beaudet, Joseph R Ecker, et al. The nih roadmap epigenomics mapping consortium. *Nature biotechnology*, 28(10):1045–1048, 2010.

[15] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[16] Jason D Buenrostro, Beijing Wu, Howard Y Chang, and William J Greenleaf. Atac-seq: A method for assaying chromatin accessibility genome-wide. *Current protocols in molecular biology*, pages 21–29, 2015.

[17] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

[18] Tsu-Pei Chiu, Federico Comoglio, Tianyin Zhou, Lin Yang, Renato Paro, and Remo Rohs. Dnashaper: an r/bioconductor package for dna shape prediction and feature encoding. *Bioinformatics*, 32(8):1211–1213, 2016.

[19] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*, 2015.

[20] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[21] ENCODE Project Consortium et al. The encode (encyclopedia of dna elements) project. *Science*, 306(5696):636–640, 2004.

[22] Fantom Consortium et al. A promoter-level mammalian expression atlas. *Nature*, 507(7493):462–470, 2014.

[23] Kairong Cui and Keji Zhao. Genome-wide approaches to determining nucleosome occupancy in metazoans using mnase-seq. *Chromatin Remodeling: Methods and Protocols*, pages 413–419, 2012.

[24] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[25] Li Fei-Fei. Imagenet: crowdsourcing, benchmarking & other cool things. In *CMU VASC Seminar*, 2010.

[26] Paul G Giresi, Jonghwan Kim, Ryan M McDaniell, Vishwanath R Iyer, and Jason D Lieb. Faire (formaldehyde-assisted isolation of regulatory elements) isolates active regulatory elements from human chromatin. *Genome research*, 17(6):877–885, 2007.

[27] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[28] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.

[29] Leif Groop. Open chromatin and diabetes risk. *Nature genetics*, 42(3), 2010.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[32] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

[33] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[34] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[35] David S Johnson, Ali Mortazavi, Richard M Myers, and Barbara Wold. Genome-wide mapping of in vivo protein-dna interactions. *Science*, 316(5830):1497–1502, 2007.

[36] David R Kelley, Jasper Snoek, and John L Rinn. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome research*, 26(7):990–999, 2016.

[37] W James Kent, Ann S Zweig, G Barber, Angie S Hinrichs, and Donna Karolchik. Bigwig and bigbed: enabling browsing of large distributed datasets. *Bioinformatics*, 26(17):2204–2207, 2010.

[38] Tae Hoon Kim, Ziedulla K Abdullaev, Andrew D Smith, Keith A Ching, Dmitri I Loukinov, Roland D Green, Michael Q Zhang, Victor V Lobanenkov, and Bing Ren. Analysis of the vertebrate insulator protein ctcf-binding sites in the human genome. *Cell*, 128(6):1231–1245, 2007.

[39] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[40] Rimantas Kodzius, Miki Kojima, Hiromi Nishiyori, Mari Nakamura, Shiro Fukuda, Michihira Tagami, Daisuke Sasaki, Kengo Imamura, Chikatoshi Kai, Matthias Harbers, et al. Cage: cap analysis of gene expression. *Nature methods*, 3(3):211–222, 2006.

[41] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[42] Eric S Lander, Lauren M Linton, Bruce Birren, Chad Nusbaum, Michael C Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[43] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[44] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, et al. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[45] Qunhua Li, James B Brown, Haiyan Huang, and Peter J Bickel. Measuring reproducibility of high-throughput experiments. *The annals of applied statistics*, pages 1752–1779, 2011.

[46] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[47] Philip Machanick and Timothy L Bailey. Meme-chip: motif analysis of large dna datasets. *Bioinformatics*, 27(12):1696–1697, 2011.

[48] Anthony Mathelier, Beibei Xin, Tsu-Pei Chiu, Lin Yang, Remo Rohs, and Wyeth W Wasserman. Dna shape features improve transcription factor binding site predictions in vivo. *Cell systems*, 3(3):278–286, 2016.

[49] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/k2). In *Doklady an SSSR*, volume 269, pages 543–547, 1983.

[50] Ryan Poplin, Dan Newburger, Jojo Dijamco, Nam Nguyen, Dion Loy, Sam S Gross, Cory Y McLean, and Mark A DePristo. Creating a universal snp and small indel variant caller with deep neural networks. *bioRxiv*, page 092890, 2016.

[51] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[52] Daniel Quang and Xiaohui Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *Nucleic acids research*, 44(11):e107–e107, 2016.

[53] James T Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S Lander, Gad Getz, and Jill P Mesirov. Integrative genomics viewer. *Nature biotechnology*, 29(1):24–26, 2011.

[54] Albin Sandelin, Wynand Alkema, Pär Engström, Wyeth W Wasserman, and Boris Lenhard. Jaspar: an open-access database for eukaryotic transcription factor binding profiles. *Nucleic acids research*, 32(suppl 1):D91–D94, 2004.

[55] Jacob Schreiber, Maxwell Libbrecht, Jeffrey Bilmes, and William Noble. Nucleotide sequence and dnasei sensitivity are predictive of 3d chromatin architecture. *bioRxiv*, page 103614, 2017.

[56] Toshiyuki Shiraki, Shinji Kondo, Shintaro Katayama, Kazunori Waki, Takeya Kasukawa, Hideya Kawaji, Rimantas Kodzius, Akira Watahiki, Mari Nakamura, Takahiro Arakawa, et al. Cap analysis gene expression for high-throughput analysis of transcriptional starting point and identification of promoter usage. *Proceedings of the National Academy of Sciences*, 100(26):15776–15781, 2003.

[57] Avanti Shrikumar, Peyton Greenside, Anna Y Shcherbina, and Anshul Kundaje. Not just a black box: Interpretable deep learning by propagating activation differences.

[58] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[60] Ritambhara Singh, Jack Lanchantin, Gabriel Robins, and Yanjun Qi. Deepchrome: deep-learning for predicting gene expression from histone modifications. *Bioinformatics*, 32(17):i639–i648, 2016.

[61] Lingyun Song and Gregory E Crawford. Dnase-seq: a high-resolution technique for mapping active gene regulatory elements across the genome from mammalian cells. *Cold Spring Harbor Protocols*, 2010(2):pdb–prot5384, 2010.

[62] François Spitz and Eileen EM Furlong. Transcription factors: from enhancer binding to developmental control. *Nature Reviews Genetics*, 13(9):613–626, 2012.

[63] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[64] John A Stamatoyannopoulos. Illuminating eukaryotic transcription start sites. *Nature methods*, 7(7):501–503, 2010.

[65] Gary D Stormo. Dna binding sites: representation and discovery. *Bioinformatics*, 16(1):16–23, 2000.

[66] Gary D Stormo, Thomas D Schneider, Larry Gold, and Andrzej Ehrenfeucht. Use of the perceptronalgorithm to distinguish translational initiation sites in e. coli. *Nucleic acids research*, 10(9):2997–3011, 1982.

[67] William R Swindell, Andrew Johnston, Liou Sun, Xianying Xing, Gary J Fisher, Martha L Bulyk, James T Elder, and Johann E Gudjonsson. Meta-profiles of gene expression during aging: limited similarities between mouse and human and an unexpectedly decreased inflammatory signature. *PloS one*, 7(3):e33204, 2012.

[68] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[69] Zhong Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature reviews genetics*, 10(1):57–63, 2009.

[70] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[71] Haoyang Zeng, Matthew D Edwards, Ge Liu, and David K Gifford. Convolutional neural network architectures for predicting dna–protein binding. *Bioinformatics*, 32(12):i121–i127, 2016.

[72] Daniel R Zerbino, Nathan Johnson, Thomas Juettemann, Steven P Wilder, and Paul Flicek. Wiggletools: parallel processing of large collections of genome-wide datasets for visualization and statistical analysis. *Bioinformatics*, page btt737, 2013.

[73] Yong Zhang, Tao Liu, Clifford A Meyer, Jérôme Eeckhoute, David S Johnson, Bradley E Bernstein, Chad Nusbaum, Richard M Myers, Myles Brown, Wei Li, et al. Model-based analysis of chip-seq (macs). *Genome biology*, 9(9):R137, 2008.

[74] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934, 2015.

[75] Tianyin Zhou, Lin Yang, Yan Lu, Iris Dror, Ana Carolina Dantas Machado, Tahereh Ghane, Rosa Di Felice, and Remo Rohs. Dnashape: a method for the high-throughput prediction of dna structural features on a genomic scale. *Nucleic acids research*, page gkt437, 2013.