Master Thesis

Self-supervised Representation Learning for Genome Sequence Data

Xiao-Yin Janet To

Supervisors: Dr. Mina Rezaei, Martin Binder
Date: December 23^{rd}, 2021

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others.

Munich, December 23^{rd}, 2021

.....
Xiao-Yin Janet To

Abstract

One major challenge for machine learning in genomics is the scarcity of labeled data. In order to obtain high quality labels, it is often necessary to perform expensive experiments. However, in the age of high throughput sequencing, there is a large quantity of unlabeled data that can be used to aid classification through semi-supervised learning. Out of the multiple possible avenues to semi-supervised learning, we tackle the problem through representation learning, which has shown promising results in previous studies. We investigate this issue using representations for semi-supervised-learning by applying a Contrastive Predictive Coding architecture and a language model predicting the next following nucleotide from a given genome sequence. The representations are evaluated by three downstream tasks from genomics: (1) The differentiation between Gram-positive and Gram-negative bacteria, (2) the differentiation between bacteria, phage-viruses, and other viruses, and (3) the recognition of different chromatin effects of sequence alterations in the human genome. The performance is compared with fully supervised approaches. We find that the language model predicting the next nucleotide from a sequence performs better than the CPC model. However, the gain in performance compared to the fully supervised approaches is rather small and further research is needed to draw more definitive conclusions.

Contents

Chapter 1

Introduction and Motivation

1.1 Genome Sequences

Just as human beings use languages to communicate, nature created its own language: genomes. Using structures invisible to the human eye, inter- and intracellular exchanged of information enables the existence life as we know it. In order to understand this "language of life", Natural Language Processing methods have been used with the aim of understanding these invisible structures [Asgari and Mofrad, 2015].

A genome is the entirety of all genetic information of a cell, stored as sequences of nucleotides on double-stranded DNA molecules. A distinction is made between coding and non-coding strands, with coding sections on the DNA being responsible for the production of RNA and proteins. The latter are the main contributors to the complex function and structure of a cell, wherefore the identification and comprehension of protein-coding genes is essential and valuable for decoding genomes and predicting their functions [Harrow et al., 2009, Szalai et al., 2020, Alberts, 1998]. Thanks to scientific progress, genomes are nowadays being sequenced at tremendous speed and multiple databases are offering a considerable collection of sequences [Sayers et al., 2020]. Simultaneously it should be noted that these sequences are to a large extent not annotated. While sequencing itself can be realized in an automated manner using scientific instruments, the identification of a single protein's functions needs to be performed manually by expensive experiments that require human annotators with high biological expertise to be performed. Additionally, they can only be carried out in clinical microbiological laboratories, using specialized software, and each experiment only allows a limited number of samples. Further, not all labeling strategies can be applied to all sequences, which makes it difficult to speed up the process by labeling all of the sequences at once [Joensen et al., 2014, Neilson et al., 2011].

Given the amount of available data, the annotation of all genomes using these

procedures might require decades of experimenting. As the utilization of computational methods have sped up the process of sequencing, we aim to speed up the process of annotation correspondingly.

1.2 Self-Supervised Representation Learning

For classification tasks, the majority of current research uses supervised learning methods using only labeled data for the training of a model. As there are many domains where labeled data is expensive to acquire and therefore not available in a large amount, a large part of current research effort is focusing on methods that do not require a large amount of expensive supervision [Kolesnikov et al., 2019]. We apply semi-supervised learning, an approach that allows the usage of both unlabeled and labeled samples by exploiting the quantity of available unlabeled genome data in comparison to the amount of data that is labeled. Semi-supervised learning has proven powerful for instances where large amounts of unlabeled data is available, as these are used for learning in addition to the supervised data [Chapelle et al., 2006]. The basic procedure we focus on in this work is self-supervised semi-supervised learning (S^4L) and can be divided into two parts: At first, only unlabeled data is used to train a model on a task that can be computed without supervision, creating representations of the data. These representations are then used as a base for training a supervised model on the labeled data by feeding them to the model as input features. In order to realize the first step, the pretraining step, a self-supervised model is trained by defining a so-called pre-text task, which is a task formed by the method, that does not require labeled data, but high-level semantic understanding in order to be solved [Zhai et al., 2019]. This model creates so-called "representations", which are activations of hidden neurons representing relevant features of the data and are expected to contain the data's essential information in a compressed and interpretable manner, making an increase of the learning algorithms' performance possible. The advantage of the representations learned by representation learning is their capability to capture the underlying factors of the data, which can be relevant for the task that is later evaluated in the supervised step [Asgari and Mofrad, 2015, Bengio et al., 2014]. For instance, recent work has shown that in the text domain useful representations could be learned by training a model that predicts a word given its context, which complies with the surrounding words in this case [Doersch et al., 2016, Mikolov et al., 2013].

1.3 Purpose and Research Question

The objective of this thesis is to find out whether S^4L approaches lead to a better performance in genome sequence annotation than fully supervised methods. For the

self-supervised step, Contrastive Predictive Coding as presented by van den Oord et al. [2019] and Hénaff et al. [2020] is compared to a language model predicting the next nucleotide given a sequence. This is evaluated by comparing the results for three downstream classification tasks: The first one being the differentiation between Gram-positive and Gram-negative bacteria, the second between bacteria, viruses that attack bacteria ("bacteriophages"), and other viruses (in the following just "viruses"), and the third one the recognition of different chromatin effects of sequence alterations in the human genome. The tasks, including the underlying data will be further explained in Chapter 3.2. Additionally, within this framework we want to evaluate if, and to what degree, performance is influenced by specific settings such as model depth or learning rate and if the representations resulting from the self-supervised pretraining are themselves interpretable.

Chapter 2

Background and Related Work

2.1 Deep Learning

Deep learning based methods have recently achieved breakthroughs in bioinformatics by exceeding the performance of previous state-of-the-art approaches [Zhang et al., 2021], for example in disease detection such as the prediction of cancer recurrence through gene expressions [Tang et al., 2019], or in drug discovery by predicting the drug-target interaction [Chen et al., 2018, Rifaioglu et al., 2018]. To develop a model capable of learning useful representations from genome sequence data, Trabelsi et al. [2019] explored several natural language processing approaches, and in the course of this the use of hybrid models combining recurrent and convolutional neural networks proved most promising. In the following, we will therefore examine the "DanQ" architecture [Quang and Xie, 2016], serving as a well established hybrid architecture for genome sequence data. Furthermore, since residual convolutional neural networks were appointed to be high-quality learners for representations [Kolesnikov et al., 2019], we investigate their application to the dataset.

2.1.1 Convolutional Neural Networks

Early versions of convolutional neural networks (CNNs) were introduced by LeCun et al. [1995], who used them for creating a method for recognizing handwritten digits. The intuition behind CNNs is to capture local patterns in data using shared weights, where filters extract features within their receptive field along the input and pass the result to a non-linear activation function σ. In the one-dimensional case, the main parameters of a convolutional layer are the input length i_l, the number of input channels i_c, the size of the receptive field of the filters $W^k = M \times i_c$ with window size M, and the output with length o_l and channels o_c. The procedure of a

convolutional layer can be described as follows:

$$convolution(X)_{j,k} = \sigma(\sum_{m=0}^{M-1}\sum_{n=0}^{i_c-1} W_{mi_c}^{k} X_{j+m,i_c}), \tag{2.1}$$

where the index of the filter is denoted by k, j denotes the index of the output position [Trabelsi et al., 2019], and the size of $convolution(X)$ is a $o_l \times o_c$ matrix.

Since our input of interest, genome sequence data, is one-dimensional, one-dimensional convolutional layers are used. Within this context, the first convolutional layer takes a one-hot-encoded input sequence of length i_l as its input, represented by a $i_l \times 4$ matrix, as there are four possible letters for nucleotides (A, C, G, T). Subsequent layers follow the same procedure and take the outputs of previous layers as an input, having a different channel size respectively. In case of multiple successive convolutional layers the first layers detect low-level features and deeper layers can identify high-dimensional global information as their receptive field is larger. This allows for the detection of interactions between positions at greater distances from each other within the genome within these deeper layers.

2.1.2 Residual neural network

Kolesnikov et al. [2019] have proposed that deep networks of many CNN layers trained to solve pre-text tasks encode high-level semantic representations useful for solving downstream tasks of interest. This makes models with deep architectures such as residual convolutional neural networks (ResNets) the architecture of choice. The idea suggested by He et al. [2015] was to create a method that simplifies the training of deeper neural networks by reformulating the layers as learning residual functions, while preventing the vanishing gradient problem [Glorot and Bengio, 2010] that often occurs when training deep networks. Residual functions use "identity shortcut connections" that skip one or more layers, as shown in Figure 2.1, instead of training a model to fit a desired underlying mapping every few layers. The mapping is formulated by $\mathcal{F}(X) + X$ and adds the outputs of the identity X to the outputs of the stacked layers $\mathcal{F}(X)$ without adding any additional parameters or extra computational complexity. A building block with input X, output y, and weights W is defined as

$$y = \mathcal{F}(X, \{W_i\}) + X \tag{2.2}$$

with i being the number of layers within the block. Each block consists of a repeating pattern of a convolutional layer, a subsequent batch normalization, rectified linear (ReLU) activation, and a shortcut connection. The typical ResNet architecture starts with a plain convolutional layer followed by a pooling layer, then four residual stacks are deployed in which multiple block architectures are assembled. The number of repetitions within each block differs between architectures. Ultimately, a global

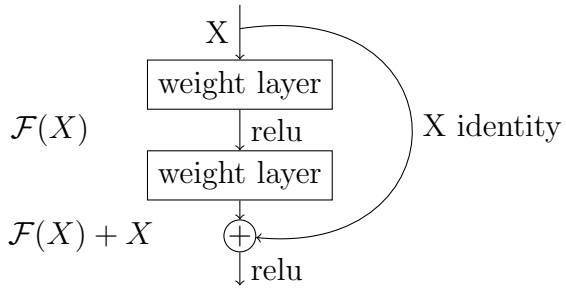

Figure 2.1: Residual learning: a building block. Reproduced from He et al. [2015]. X denotes the input to a building block. Through a shortcut connection by identity mapping (X identity) fits a residual mapping, producing $\mathcal{F}(X) + X$ at the end of each building block.

average pooling layer reduces the number of parameters to be trained, and a fully connected layer with softmax activation produces the final prediction.

2.1.3 Recurrent neural networks

Early versions of recurrent neural networks (RNNs) were based on the concepts acquired by Hopfield [1982], who introduced the "Hopfield network" that was designed to simulate human memory, and Rumelhart et al. [1986], who addressed sequential data by networks with loops, allowing information to persist through back-propagation of the error. This approach is mainly used for sequential data $x_1, \ldots, x_n$, as it is able to memorize previous inputs and take them into consideration when determining the output. To reach this objective, units are connected in the form of a directed cycle, and the model includes additional state variables that store past information in order to feed them to the next time frame in the network along with the present information, each connection having its own trainable parameters. However, when attempting to learn long-term dependencies, the simple implementation suffers from vanishing or exploding gradients.

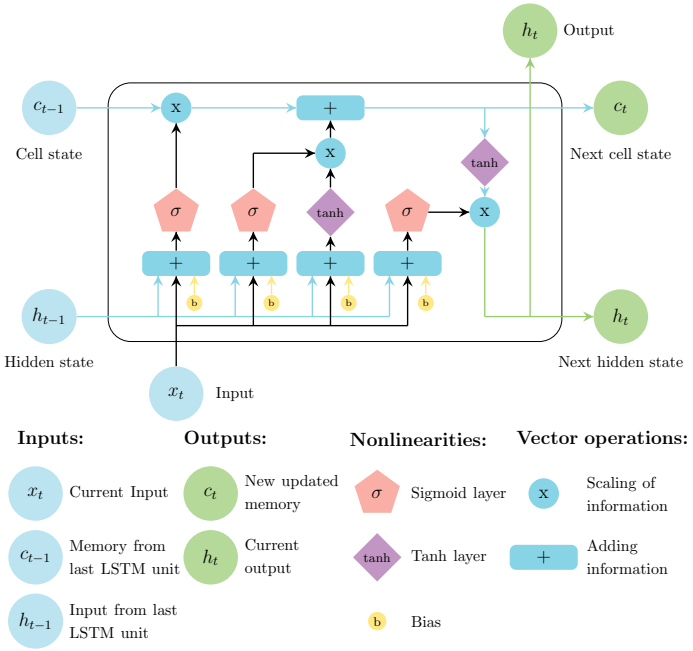

Figure 2.2: The structure of the Long Short-Term Memory (LSTM) neural network. Reproduced from Le et al. [2019].

While there are a variety of different types of RNNs, in this work we focus on Long Short-Term Memory (LSTM) networks [Hochreiter and Schmidhuber, 1997], which are designed to tackle the obstacle of learning long-term dependent correlations within data and the vanishing and exploding gradients issue by exhibiting dynamic temporal or spatial behavior. The most commonly used architecture is based on the concept introduced by Hochreiter and Schmidhuber [1997] and was further developed by Gers et al. [1999]. Compared to simple RNN architectures, the LSTM architecture is extended by memory cells and three types of gate units: The input gate, which adds new information to the memory cell while preventing irrelevant information from entering, the forget gate, which is able to reset memory blocks and remove irrelevant and obsolete low-level information, and the output gate, which decides what information on previous inputs is passed to the next hidden state and the next cell state. A detailed overview of the functionalities of this architecture is displayed in Figure 2.2.

2.1.4 Hybrid models

Hybrid models based on CNNs and RNNs have become the current state-of-the-art architectures for genome sequence modeling [Alipanahi et al., 2015, Hassanzadeh and Wang, 2016]. Assembling both network types, the model is able to identify sequential and structural motifs. While CNN layers can learn features within the sequence,

RNN layers improve performance by learning long-term dependencies [Trabelsi et al., 2019].

An example of a well established hybrid architecture combining RNN and CNN layers is the "DanQ" architecture introduced by Quang and Xie [2016], which was originally applied for DNA sequence function prediction using the dataset from the DeepSEA framework [Zhou and Troyanskaya, 2015]. The input layer takes the one-hot-encoded vector of a sequence and passes it to a convolutional layer with ReLU activation. Following this, a max pooling layer reduces the sequence length, which is then relayed to a bidirectional LSTM layer. The output from this layer is flattened and input to a fully connected hidden layer, followed by a fully connected output layer, which returns the predictions as a vector of length 919 after a sigmoid transformation, as their objective is a multi-task prediction with 919 target labels.

2.1.5 Optimization and learning rate

Stochastic gradient-based optimization is at the core of many high-dimensional optimization processes in numerous areas of science as well as industry and engineering. Within this scope, methods based on first-order derivatives such as the stochastic gradient descent are advantageous compared to methods based on higher-order derivatives, which would incur high efficiency combined with low computational complexity, since higher-order derivatives are associated with a higher usage of computational memory.

Adam

One method that demonstrated notable improvements in optimizing stochastic objectives with high-dimensional parameter spaces is the Adaptive Moment Estimation (Adam) method [Kingma et al., 2014], which requires only first-order gradients of the objective function and thus relatively little memory, while it even performs well for sparse gradients or on-line and non-stationary settings and is invariant to rescaling of the gradient. Here, the trainable parameters θ_t are updated in each epoch by adapting to the exponentially decaying average of past gradients and past squared gradients, which correspond to the first moment m_0 and second moment v_0. The parameters $\beta_1, \beta_2 \in [0, 1)$ denote the exponential decay rates of the moment estimates, α denotes the step size, which is equal to the learning rate and remains constant over the course of the training, and a small constant ϵ is included to prevent a division by zero in the update step. The application of the exponentially weighted average of gradients at different moments accelerate the convergence of the algorithm. The full algorithm as presented by Kingma et al. [2014] is shown in the following pseudocode in Algorithm 1, giving a clear overview of the process.

Algorithm 1 The Adam Algorithm [Kingma et al., 2014]

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: ϵ: Small constant
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize first moment vector)
  $v_0 \leftarrow 0$ (Initialize second moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Learning rate schedule: Cosine Annealing

To accelerate the rate of convergence, researchers tested different approaches to adjust the learning rate during training [Smith, 2015, Zagoruyko and Komodakis, 2016], and that process, the so-called "Stochastic Gradient Descent with Warm Restarts" (SGDR) introduced by Loshchilov and Hutter [2017], was demonstrated as a new state-of-the-art technique. Instead of the constant learning rate described for the Adam algorithm, the learning rate is here adapted for each epoch.

In this method, the learning rate is defined as a value in $[\eta_{min},\ \eta_{max}]$. Along the training, the learning rate decreases with cosine annealing for each epoch and is reset to the initial value after a defined instance of time until a restart T_i, with i defined as the time interval between two restarts. The time until a restart T_i can change over the training by increasing it by a factor T_{mult}. Loshchilov and Hutter [2017] suggest setting T_i to a small value at the beginning of training and increasing after each restart to improve anytime performance. Finally, the learning rate η_t for epoch t and iteration i is computed as follows:

$$\eta_t = \eta_{min} + \frac{1}{2}\left(\eta_{max} - \eta_{min}\right)\left(1 + cos\left(\frac{T_{cur}(t)}{T_i(t)}\pi\right)\right). \tag{2.3}$$

The process of this learning rate schedule is visualized by five example settings in Figure 2.3.

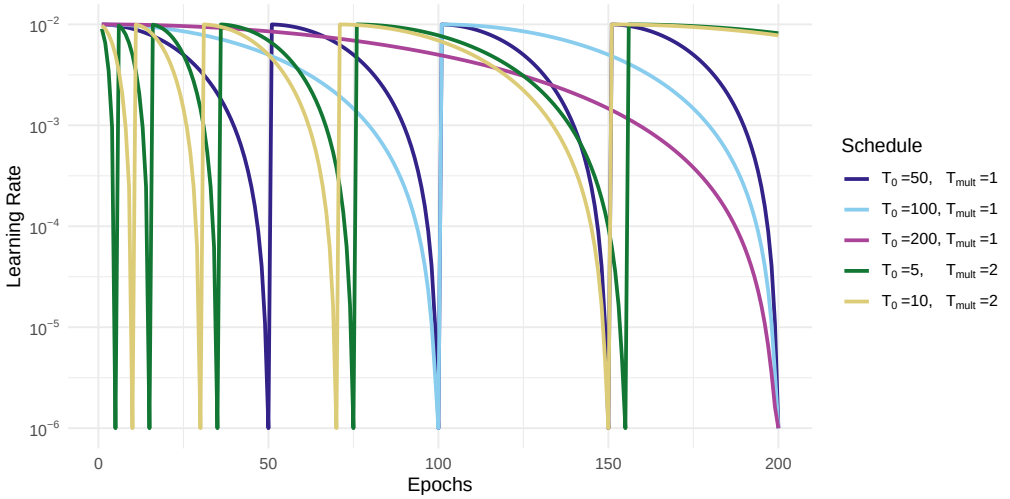

Figure 2.3: Cosine Annealing schedule scheme of learning rate η_t over batch index t. Warm restarts simulated every $T_0 = 50$ (dark blue line), $T_0 = 100$ (blue line) and $T_0 = 200$ (pink line) epochs with η_t decaying during i-th run from $\eta^i_{max} = 0.01$ to $\eta^i_{min} = 10^{-6}$ according to Eq. (2.3); warm restarts starting from epoch $T_0 = 5$ (dark green line) and $T_0 = 10$ (yellow line) with doubling ($T_{mult} = 2$) periods T_i at every new warm restart.

2.2 Deep Representation Learning Techniques

2.2.1 Self-supervised learning Semi-supervised learning

The term semi-supervised learning refers to all approaches that tackle the the problem of learning a machine learning model when only a small number of labeled samples and large number unlabeled samples samples are available by using both labeled and unlabeled samples for learning, in most instances applied to classification tasks. Within the framework of this thesis, the focus is on semi-supervised methods based on deep neural networks [Zhai et al., 2019, Chapelle et al., 2006]. The challenge here is to make use of the unlabeled samples and learn properties of the data that can be used by a classification model, with the goal of improving accuracy compared to a model trained solely on supervised data. For this purpose, the aim is to learn not only features that are useful for a specific supervised task, but features that transfer well to related domains.

Many of the early implementations used generative models as a foundation [Rasmus et al., 2015, Odena, 2016, Adiwardana et al., 2016] because generative approaches require a high-level understanding of the structure of the data in order to model it. Many experiments have been reported where the quantity of labeled components could be significantly reduced using generative techniques [Zhu, 2005].

Due to the empirical success acquired in both academia and industry, another foundation called representation learning has gained attention in recent years [Bengio et al., 2014, Szummer and Jaakkola, 2002, Chapelle, 2003]. The given unlabeled samples are used for the creation of data representations, which are activations of hidden neurons representing relevant features of the data and are expected to contain the data's underlying factors in a compressed manner, and to be interpretable themselves, or at least by a trained supervised downstream model. These are fed into a supervised classifier with the goal of capturing only the relevant data structures and characteristics rather than the full raw data, which may contain information that does not aid to solve the task. Notable success in representation learning have been reported in signal processing [Boulanger-Lewandowski et al., 2012], object recognition [Doersch et al., 2016], and natural language processing [Kawakami et al., 2020], among others. Because representations can enclose information useful for multiple tasks with common factors, they have also been found to be advantageous in the context of transfer learning and multi-task learning [Krizhevsky et al., 2012].

One promising approach for creating representations is self-supervised learning, a subclass of unsupervised learning. While in generative models the data needs to be understood sufficiently to permit reconstruction, here the data is used directly to formulate a so-called pre-text task [Zhai et al., 2019, Hénaff et al., 2020]. In this process, pseudo-labels are generated from the unlabeled data to enable the execution of supervised methods. Common pre-text tasks are the prediction of future, missing, or contextual information. For instance, in image processing, a possible pre-text task is to predict the angle of rotation transformation used to modify an input image as shown by Gidaris et al. [2018]. An example from the field of reinforcement learning teaches a robot to learn motions seen in an image by mimicking them and comparing them to its own mirror reflection [Sermanet et al., 2017].

Outperforming previous state-of-the-art methods, influential models that apply self-supervision include BERT, introduced by Devlin et al. [2018] in the natural language domain, and SimCLR introduced by Chen et al. [2020] as well as the unsupervised learning of visual representations by context prediction approach proposed by Doersch et al. [2016] in the image domain. The latter proposed using patch-based techniques, such as predicting a patch given its context as a pre-text task, which inspired many scientists to use patch-based techniques in numerous other domains as well. Notable success has been achieved in PASCAL VOC 2007 classification, an object recognition task with a set of visual object classes in realistic scenes, where patch-based self-supervised representation learning has become the new state-of-the-art method [Doersch et al., 2016, Hénaff et al., 2020], and moreover in the field of bioinformatics, where Lu et al. [2020] have successfully trained representations of protein sequences and evaluated them on several downstream tasks such as protein structure classification or protein stability prediction.

2.2.2 Contrastive predictive coding

Another patch-based self-supervised technique that has attracted considerable attention was *Representation Learning with Contrastive Predictive Coding* (CPC) introduced by van den Oord et al. [2019] and further developed by Hénaff et al. [2020]. The idea of predictive coding was inspired by early signal processing techniques for data compression as proposed by Atal and Schroeder [1978], which in turn was inspired by the abstraction of observations employed by the brain, that can be inspected in neuroscience research [Friston, 2005]. Just as the brain does not store all perceived details and instead retains only the general framework, also for representations, the abstraction should be accomplished through selecting only the underlying shared information between different parts of the high-dimensional signal and global structures for the encoding, while discarding local noise and less relevant information. This type of high-dimensional global information are called "slow features". For instance, in order to solve a classification task, the ability to recognize a cat in an image is more relevant than memorizing the exact color and position of its whiskers. Although this can be effortlessly realized by human beings, this global structure is far more difficult for a computer to recognize, since pixels must be analyzed in light of their context. Van den Oord et al. [2019] state that it is context based approaches that are able to learn useful representations, because a large number of possible targets are conditionally dependent on the same high-level latent information that can be found in the data. Consequently, a method that can detect slow features is required and this challenge is addressed by modeling the target x in consideration of the context c by mapping both into distributed vector representations with a much more compact embedding space, and by preserving the mutual information through maximization. This maximized mutual information contains the extracted underlying latent variables the inputs have in common, and is defined as

$$I(x;c) = \sum_{x,c} p(x,c)\, log\, \frac{p(x|c)}{p(x)} \tag{2.4}$$

with x as the future observation of interest and c as the present context observations. This proposed method can be applied on data with, for example, spatially or temporally ordered observation and proceeds in three steps: At first, the input data x_t is compressed by a non-linear "encoder network" g_{enc} into a compact latent embedding space that forms the latent representations $z_t = g_{enc}(x_t)$. Summarized information of the context of a patch is given from autoregressive layers g_{ar} of the "context network" modeled on top, producing the latent context representation $c_t = g_{ar}(z_{\leq t})$. Both representations z_t and c_t are suitable as representations for downstream tasks, with c_t being especially recommended for targets that rely on past information [van den Oord et al., 2019].

Hénaff et al. [2020] have implemented a modified version of the CPC architecture that focuses on image classification and was found to improve the performance achieved by the initial CPC architecture. This modified version is named CPC v2, the initial architecture is thus named CPC v1. In this construction, an image is first divided into overlapping patches $x_{i,j}$ where i and j indicate the position of the patch. All patches are independently encoded by g_{enc}, resulting in representations $z_{i,j} = g_{enc}(x_{i,j})$, which are transformed into context vector representations $c_{i,j} = g_{ar}(\{z_{u,v}\}_{u \leq i,v})$. The pre-text task formulated here is to predict future feature vectors $z_{i+k,j}$ given $k > 0$ and present context vectors $c_{i,j}$. The trainable prediction matrix W_k is multiplied by the current context vector, yielding the prediction of the future feature vector

$$\hat{z}_{i+k,j} = W_k c_{i,j} \tag{2.5}$$

W_k includes the patch to be predicted as the positive sample $z_{i+k,j}$, as well as negative samples z_l from other patches from the same image and patches from other images from the same batch, that form the negative samples. Since the representations are the output of neural networks, the objective is to train a model that obtains the highest value at the position of $z_{i+k,j}$ after matrix multiplication between W_k and $c_{i,j}$ while producing values close to zero at all other positions z_l. The result depicts the probability to be assigned to a target by applying the softmax function and is assessed using cross-entropy loss. In conclusion, the CPC objective is defined as

$$\begin{aligned} \mathcal{L}_{CPC} &= -\sum_{i,j,k} log\, p\left(z_{i+k,j} \mid \hat{z}_{i+k,j}, \{z_l\}\right) \\ &= -\sum_{i,j,k} log \frac{exp(\hat{z}_{i+k,j}^T z_{i+k,j})}{exp(\hat{z}_{i+k,j}^T\, z_{i+k,j}) + \sum_l exp(\hat{z}_{i+k,j}^T\, z_l)} \end{aligned} \tag{2.6}$$

The loss is called InfoNCE, inspired by Noise-Contrastive Estimation [Gutmann and Hyvärinen, 2010, van den Oord et al., 2019].

The architecture proposed by Hénaff et al. [2020] takes images with a resolution of 260×260 pixels as input, extracts patches of the size 80×80 pixels with a stride of 36×36 pixels, amounting to a grid of 6×6 image patches. These are passed to the encoder g_{enc}, for which a modified ResNet-101 was chosen. This version includes only the first three stacks, whereas the third residual stack is extended to 4,096-dimensional feature maps and 512-dimensional bottleneck layers, instead of the 1,024-dimensional feature maps and 256-dimensional bottleneck layers introduced for the original ResNet-101 architecture by He et al. [2015]. This ResNet architecture used by Hénaff et al. [2020] is named "ResNet-161". The output of the encoder network are [6, 6, 4,096] tensor representations for each image, designated as "latents". These are aggregated through the context network into a 6×6 grid of context vectors for which a PixelCNN decoder [van den Oord et al., 2016] was chosen. In order to collect information about the context of a patch, the network

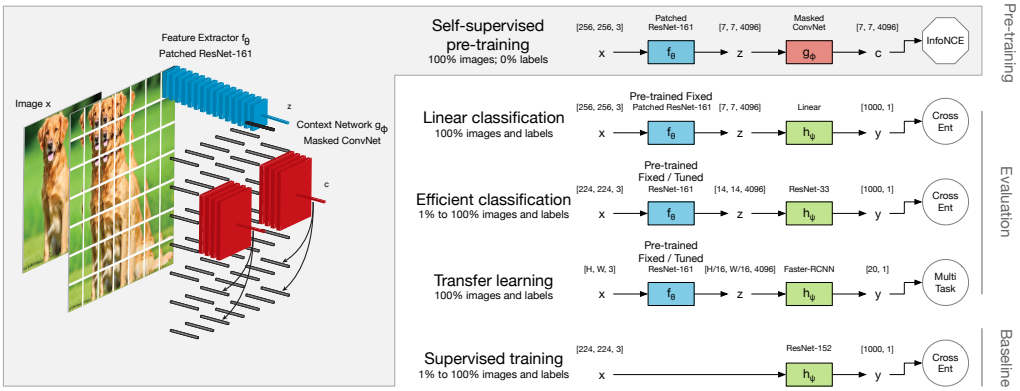

Figure 2.4: Overview of the framework for semi-supervised learning with Contrastive Predictive Coding for Image Data. Left: unsupervised pretraining with the spatial prediction task. First, an image is divided into a grid of overlapping patches. Each patch is encoded independently from the rest with a feature extractor (blue) which terminates with a mean-pooling operation, yielding a single feature vector for that patch. Doing so for all patches yields a field of such feature vectors (wireframe vectors). Feature vectors above a certain level (in this case, the center of the image) are then aggregated with a context network (red), yielding a row of context vectors which are used to linearly predict features vectors below. Right: using the CPC representation for a classification task. Having trained the encoder network, the context network (red) is discarded and replaced by a classifier network (green) which can be trained in a supervised manner. Figure taken from Hénaff et al. [2020] with permission.

contains a convolutional layer with a kernel size of [1, 3] to gather information from the horizontal context, and another convolutional layer with a kernel size of [2, 1] to gather information from the vertical context. This is repeated five times to iteratively collect information of each surrounding patch for each patch using a residual connection between the initial latents and the respective tensor yielded from each iteration, with higher importance given to closer patches. The final result is shrunk by the factor of an "embedding scale" to prevent local overfitting. The 6×6 grid of context vectors is then assessed along with the 6×6 grid of latents through the InfoNCE loss function (2.6). The process is visualized in Figure 2.4. An Adam optimizer [Kingma et al., 2014] with a learning rate of 0.0004, $\beta_1 = 0.8$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and Polyak averaging with a decay of 0.9999 was applied for optimization.

Chapter 3

Method and Experimental Setup

The S^4L architecture for genome sequence classification applied in this thesis consists of two central components. First, self-supervised training takes place where representations are generated. For this purpose, we compare two pre-text tasks: A model that predicts the patch of a sequence analogous to the CPC model and another language model that predicts the next nucleotide given a sequence. Language models are statistical models applied on language processing and can be based on words or characters. Their goal is the understanding of structures in language data and the typical task for training is the prediction of the next word or the next character given a text. Common language models are BERT introduced by Devlin et al. [2019] and GPT-3 introduced by Brown et al. [2020], among others.

The second component is supervised training, where these created representations are input for downstream task evaluation employing a supervised model trained on the labeled data. We verify the quality of the representations by evaluating the performance through linear classification via regularization using lasso regression [Tibshirani, 1996], where the lambda value for prediction is chosen by cross-validation with grid search over possible values, choosing the largest value with a cross-validation error at most one standard error larger than the smallest cross-validation error. Another approach for downstream task evaluation we used is linear classification by adding a single linear layer to the network. In the latter case, the weights of the self-supervised network creating the representations are either frozen to construct a purely linear classification model, creating the "neural network linear classification" evaluation method, or the weights remain trainable but are trained at a lower learning rate to construct a fine-tuning model, creating the "neural network fine-tuning" evaluation method. As two approaches for linear classification are examined, we refer to neural network linear classification as "NN linear classification" and to linear classification via regularization using lasso regression simply as "lasso regression". In supervised training, data are sampled balanced by target; in evaluation, all test data are used, resulting in an unbalanced

Table 3.1: Number of Samples included in one Epoch. Instead of using the entire training data in one Epoch, one Epoch was defined to include *Batch Size* $\times$ *Steps per Epoch* samples of size *Sequence Length*.

Model	Batch Size	Steps per Epoch	Sequence Length	Samples per Epoch
Self-Supervised Language Model	512	3000	500	1,536,000
Self-Supervised CPC	32	3000	6700	96,000
Supervised, Gram Staining	32	3000	6700	96,000
Supervised, Bacteria-Virus	32	3000	6700	96,000
Supervised, Chromatin Features	32	3000	1000	96,000

distribution of the target variable.

All models are trained until convergence, which is defined as the point of time, where the slope of the learning curve does not change after a restart of the cosine annealing. A time limit of two weeks is set, after which the training is terminated even if no convergence was observed yet.

3.1 Technical Aspects

Since the available data consists of about 4.5 million samples when an input sequence of length 6,700 is chosen, the duration of a single epoch would be too long to dynamically observe the training progression. Therefore, we have redefined the term "epoch" within this thesis, and in contrast to classical procedures in training a deep learning model, not all training data are considered in the course of one "epoch". Instead, the length of one input sequence, the batch size, and the number of steps per epoch are predefined in compliance with computational limits or data availability. An epoch as it was used here does consequently not contain the entire training data, but only determines after how many steps the evaluation on the validation data is triggered instead. The number of samples included in one "epoch" is summarized in Table 3.1.

Since the total size of the data amounting to 52 gigabytes exceeds the working storage of the available machines, a data generator from the deepG library [Mreches et al., 2021] that was created precisely for the preparation of genome sequence data is used. We used genome data stored in FASTA-format. FASTA-format files are files enclosing human readable text, where the first line contains the primary information of the genome and the following rows contain the sequence in ASCII-based format, represented by the four different symbols **A** for adenine, **C** for cytosine, **T** for thymine, and **G** for guanine [Wang et al., 2019]. The length of the sequence varies between different organisms from which the sequence is derived. The data is organized in folders, where a folder d contains sequence files $\{f\}$ of length l_f and

Table 3.2: Summary of the used Datasets. The exact percentages listed in the "Outcomes" column are rounded to two decimal places and refers to the exact percentage within the test dataset used for downstream task evaluation.

Dataset	Outcomes	Number of Nucleotides
Bacterial sequences	(Pre-Training Only)	39,834,931,884
Bacterial, human, and viral sequences	(Pre-Training Only)	350,829,190,515
BacDive: Bacterial sequences with Gram Staining Annotation	Gram-Positive (42.79%), Gram-Negative (57.21%)	8,138,600,419,728
Bacterial and viral sequences	Bacteria (74.47%), Phage-Virus (23.86%), Non-Phage-Virus (1.65%)	347,683,101,359
DeepSEA: Human Sequences	919 labels	4,863,024,000

only files with $l_f \geq L$ are considered, with L as the pre-specified length of a sequence input to the model. The data needed to train one batch of size n_b is sampled by a generator. In the beginning, the generator samples one f from d and takes at most $n_{\max}$ sub-sequences of length L from f. If $L \times n_b \geq l_f$, the generator samples another f' from d and extracts $L \times n_b - l_{f'}$ sequences. This continues until n_b sub-sequences are found. The remaining sub-sequences from this file are then used for the next batch and the described procedure is repeated. To ensure comparable validation measures across epochs, a validation generator with $n_{\max} = 8$ is prepared and saved for each task, providing a prediction for the same sequences in the same order at each epoch. For training the models, NVIDIA Tesla V100 GPU machines with 32 gigabytes memory capacity were used.

3.2 Data

An overview of all datasets used within the scope of this work is depicted in Table 3.2.

3.2.1 Pretraining

The data used to pretrain the self-supervised model was downloaded from GenBank [Sayers et al., 2020], a public database containing 9.9 trillion base pairs from over 2.1 billion nucleotide sequences for 478,000 formally described species. One interim finding we hope to achieve is whether higher diversity within the training data leads to the extraction of higher-level slow-features. As the primary downstream task is the classification of bacterial properties, the model was trained based on bacterial

genome sequences only on one hand, and based on a combined dataset comprising of bacterial, human, and viral sequences on the other hand. The genome database of the Helmholtz Center for Infection Research (HZI) in Braunschweig, which was extracted from GenBank, served as the data basis. The bacterial data comprises 13,264 sequences, the human data comprises 2,136 sequences, and the viral data comprises 39,109 sequences, with 25,846 being phages, viruses that infect bacteria, and 13,263 being non-phage virus.

3.2.2 Gram staining classification task

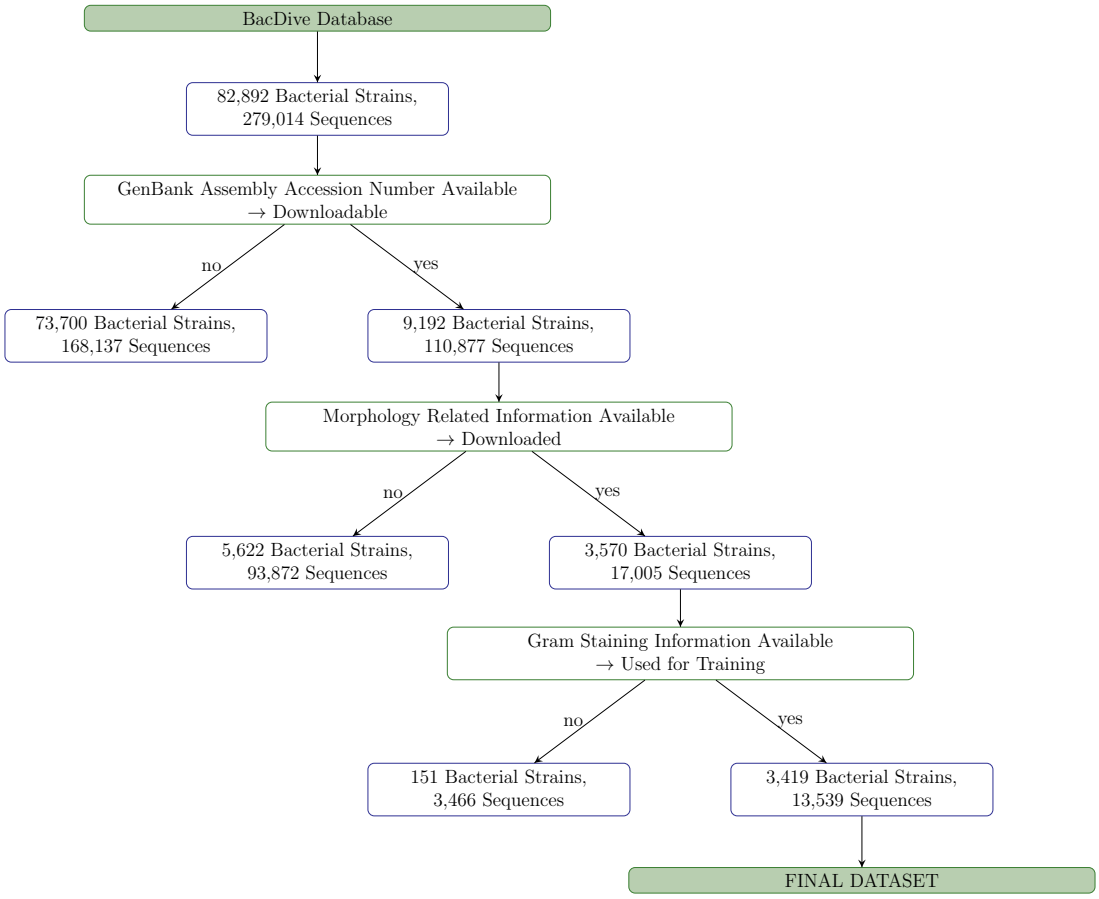

Figure 3.1: Overview of the origin of the data used for Gram staining classification

For this project, a collaboration with the Leibniz Institute DSMZ German Collection of Microorganisms and Cell Cultures collection provided us with the worldwide largest database for standardized bacterial information (BacDive) of 82,892 bacterial strains and a total of 279,014 analyzed genomes and genome parts of these

strains. The data contains a wide range of relevant information about bacteria combined from various databases maintained by biological resource centers, unpublished collections from researchers, and extracted species descriptions from the primary literature. Out of the 82,892 strains, 9,192 strains corresponding to 110,877 genomes sequences for a so-called GenBank assembly accession number is available, which makes it possible to download the corresponding genome sequence as a FASTA-file from GenBank [Söhngen et al., 2015, Reimer et al., 2018]. For the analyses within this thesis, a subset of the data comprising of 17,005 genome sequences with morphology-related data points corresponding to 3,570 strains was downloaded. The origin of the used data is visualized in Figure 3.1.

As a primary target, we chose the prediction of Gram staining properties. Gram staining is a common procedure for the classification of bacteria into two main groups that are distinguished by the chemical and physical properties of their cell wall. In laboratory for Gram staining, a bacterium is first stained with a special dye called crystal violet and then decolorized with ethanol. Gram-positive bacteria adhere the stain in their cell wall and are considerably harder to decolorize than Gram-negative bacteria. The bacterium is then counterstained by adding a red dye to gain certainty. Hereafter, Gram-positive cells remain purple, while Gram-negative cells are stained red [Hucker and Conn, 1923]. The information whether bacteria is Gram-positive or Gram-negative, among others, aids the treatment of bacterial infections, as Gram-positive infections are medicated using other antibiotics as Gram-negative infections.

The data from GenBank used for analyses include 13,539 bacterial genomes, matched to 3,419 strains with Gram-staining information. Out of these, 28% are labeled as Gram-positive and 72% are labeled as Gram-negative, a distribution that roughly corresponds to the distribution of naturally occurring bacteria [Akhtar et al., 2014, Jan et al., 2014, Sizar and Unakal, 2020].

3.2.3 Bacteria-Virus Classification task

As described in Chapter 3.2, the data we have used for pretraining contains either bacteria sequences only or sequences from bacteria, humans, and viruses with the viruses being divided into the group of phages and non-phages. For medical diagnoses, information on whether a disease is viral or bacterial is crucial to ensure appropriate treatment. Since the choice of laboratory experiments depends on this information and more specific knowledge about the respective bacteria or viruses are required for more accurate diagnoses, the distinction between these two groups is of great interest, and especially the distinction between phages and bacteria is particularly challenging. Because phages are viruses that infect bacteria, bacteria infected by phages mutate and in this process their genome sequences are adapted to these viruses, assimilating these two groups, and additionally phages can affect the virulence of bacteria, making their detection especially important for infection

research [Miao and Miller, 1999]. For this reason, another task we chose is the classification between bacteria and viruses with distinction between phage and non-phage viruses.

3.2.4 Chromatin features classification task

Zhou and Troyanskaya [2015] have created a diverse compendium of genome-wide chromatin profiles, including human genome sequences with annotations on 919 different chromatin features as part of their "DeepSEA" framework. Chromatin is the ensemble of all units of basic genetic material within the cell nucleus. The function of the chromatin is based on the DNA it contains and the challenge we address here is multitask-prediction, on one hand, and transfer learning on the other hand, since multiple annotations are available for each sequence and for some models, pretraining is conducted on bacteria genomes only, whereas the domain of interest here is humans, which are a species within the taxonomic kingdom of animalia.

3.3 Experiments

At the beginning, different learning rates are evaluated in one experiment. Based on the learning rate with the best performance, the second experiment evaluates the best value for the embedding scale, i.e., the factor by which the result of the context network is shrunk before the residual connection, as described in Chapter 2.2.2. For these two experiments, the result is evaluated by comparing the value returned by the InfoNCE loss function (2.6) and the values resulting in the lowest InfoNCE loss are kept for further analyses. The consequential learning rate is hereby the initial learning rate for training using Adam optimization [Kingma et al., 2014] with Cosine Annealing learning rate schedule Loshchilov and Hutter [2017], as in preliminary experiments the addition of this schedule has shown to accelerate convergence comparing to the plain Adam optimization. After determining the learning rate and embedding scale, we want to disclose whether the number of samples taken from a sequence affects the performance by creating the representations using a maximum of 16 or 80 samples per file for all architectures and comparing the performance on the Gram staining classification downstream task. The architecture with the best performance associated with the maximum number of samples per file is retained for further analyses, while the other settings are discarded. An overview of the applied architectures is depicted in Table 3.3.

For implementation, the R interface to Keras was used [Chollet et al., 2017]. We built the models using the functional API and trained them using a custom function instead of the pre-implemented `keras::fit()` function, since this one does not support training self-supervised models.

Table 3.3: Overview of applied architectures: Pretraining and baseline

Name	Architecture	Trainable Parameters	Dimension of Resulting Representations
S^4L-CPC-RN50	Modified 1D-CPC: 1D-ResNet-50 encoder, LSTM context network	18,489,536	2048
S^4L-CPC-RN18	Modified 1D-CPC: 1D-ResNet-18 encoder, LSTM context network	4,690,560	512
S^4L-CPC-DanQ	Modified 1D-CPC: modified DanQ encoder, LSTM context network	12,797,085	320
S^4L-NextNuc	DanQ architecture modified for next nucleotide prediction	1,510,724	320
BL-CPC	Same as best S^4L-CPC resulting from experiment 3, but no pretraining	approx. same as best S^4L-CPC	no representation
BL-NextNuc	Same as S^4L-NextNuc, but no pretraining	1,182,402	no representation

3.4 Self-supervised Model Architecture

3.4.1 Contrastive Predictive Coding for Self-supervised Genomics Data

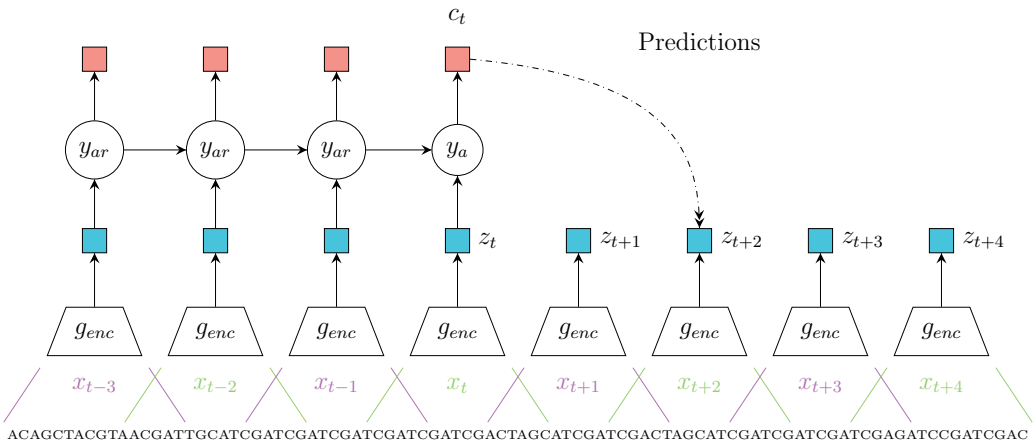

Figure 3.2: Overview of Contrastive Predictive Coding: The setup used for Genome Sequences, derived from van den Oord et al. [2019]. One patch in our architecture includes 500 nucleotides, the patches are extracted from a sequence for length 6700 with an overlap of 200 nucleotides. For a patch at position t the later patch at position $t + 2$ should be predicted. The encoder g_{enc} is a 1D ResNet-18, a 1D ResNet-50, or a modified DanQ, depending on the experimental condition.

Having achieved particularly high performance in the image domain, the encoder applied in the CPC v2 architecture combined with the initial CPC v1 architecture by van den Oord et al. [2016] is implemented in the given genome sequence domain to determine whether similarly good results can be achieved here. To match the patch-based method here, 32 patches of length 500 are extracted with stride 200 from a sequence of length 6,700, which are passed to the encoder. The batch size is set to 32. A visualization of this architecture can be found in Figure 3.2. We compare the performance of three different encoders to determine possible impact of model depth on performance. Hénaff et al. [2020] used a ResNet-161 as the encoder. Since images have a higher resolution than genome sequences, in our experiments we try three versions of encoders, using ResNet-50, ResNet-18, or an architecture close DanQ.

Table 3.4: Adapted one-dimensional ResNet architecture as CPC encoder network. The values in the brackets denote the kernel size and the channels, the value behind denotes the number of blocks in one residual stack.

Name	Output Size	18-Layer	50-Layer
Conv1D	248	$[5, 64] \times 1$	
Max Pooling	124	pool size 5, stride 2	
Stack 1	62	$\begin{bmatrix} 3, 64 \\ 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1, 64 \\ 3, 64 \\ 1, 256 \end{bmatrix} \times 3$
Stack 2	31	$\begin{bmatrix} 3, 128 \\ 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1, 128 \\ 3, 128 \\ 1, 512 \end{bmatrix} \times 4$
Stack 3	16	$\begin{bmatrix} 3, 256 \\ 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 1, 256 \\ 3, 256 \\ 1, 1024 \end{bmatrix} \times 6$
Stack 4	8	$\begin{bmatrix} 3, 512 \\ 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1, 512 \\ 3, 512 \\ 1, 2048 \end{bmatrix} \times 3$
Average Pooling	1	global	

The original ResNet architecture [He et al., 2015] has been adapted by changing the dimensions and replacing two-dimensional convolutional layers by one-dimensional convolutional layers. The concept of the one-dimensional ResNet-based encoders is displayed on Figure 3.3, and the detailed specifications are summarized in Table 3.4. Full visualizations of the ResNet-18 encoder architecture and the ResNet-50 encoder architecture are included in the Appendix A in Figures A.1 and A.2. The input of the encoder is the patch of a sequence of size 4×500, the output of the encoder is a sequence representation of size $4 \times \textit{output channels}$, which is 512 for the ResNet-18 encoder architecture, 2,048 for the ResNet-50 encoder architecture, and 320 for the DanQ encoder architecture.

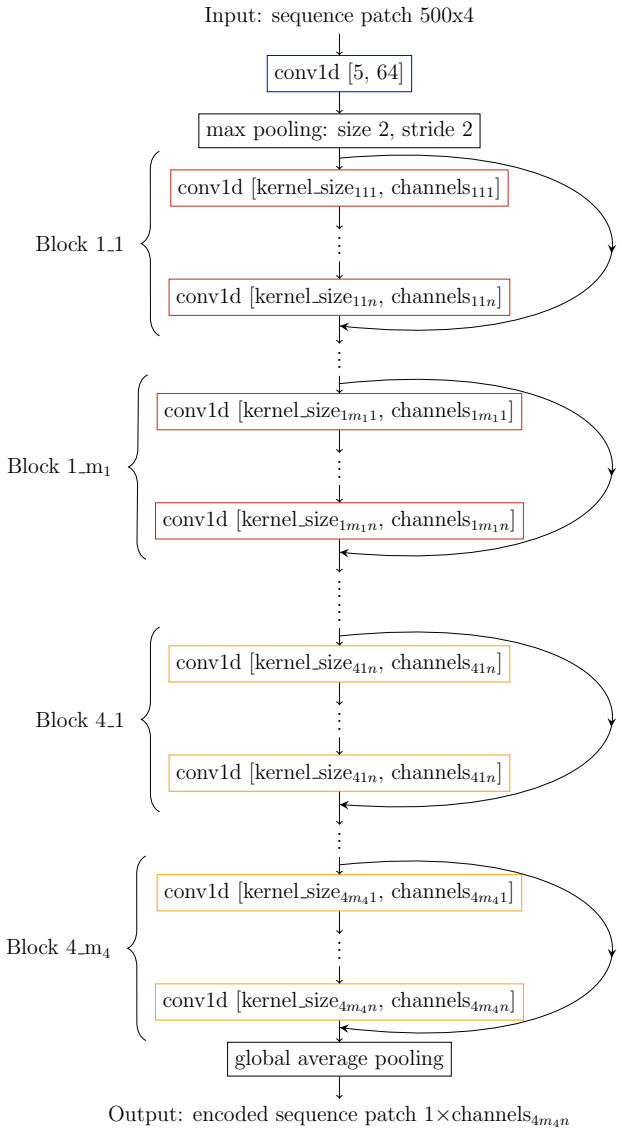

Figure 3.3: Adapted one-dimensional ResNet architecture structure as CPC encoder network. For all ResNet types, the first layer is a one dimensional convolutional layer with kernel size five and 64 channels, followed by a max pooling layer with pooling size two and stride two. The successive layers of the architecture are a construction of residual stacks, each composed of a repeating pattern of building blocks, where the number of blocks in a residual stack can vary for each stack. The number of blocks is denoted as m_i with i as the number of the stack. Within one block, the set of a one-dimensional convolutional layer, batch normalization, and ReLU activation is repeated n times, and the resulting tensor is added to the input of the block by a residual connection. ResNet types differ by the number of blocks m in one stack and the number of layers n before a shortcut connection. The output is a sequence representation.

To create a model able to perform well on the given task, we also implemented a modified version of the original DanQ architecture. Quang and Xie [2016] take the one-hot-encoded vector of a sequence as an input, pass it to a convolutional layer with ReLU activation, and reduce the size of the tensor using max pooling. While they then insert a bidirectional LSTM layer, we apply a unidirectional LSTM layer for the prediction of only subsequent patches, since the model would not profit from bidirectionality. In addition, to create the encoded patch, the final prediction layer from the original architecture is removed, so the output layer in our architecture is a dense layer with 925 channels and ReLU activation. The final architecture is displayed in Figure 3.4.

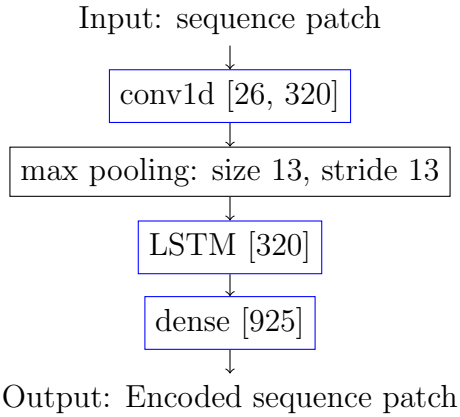

Figure 3.4: Adapted one-dimensional DanQ as CPC encoder network. The input is a sequence patch to which one dimensional convolution with kernel size 26 and 320 channels, max pooling, a LSTM layer with 320 channels returning the hidden state output for each input time-step, and finally a dense layer with 925 channels are applied. The representation of the sequence is the output of the LSTM layer.

Furthermore, Hénaff et al. [2020] have divided the image to six patches, which are then encoded and given to the iterative "PixelCNN" context network. Since there are 32 patches in our case and applying the same method would result in a large number of convolutional layers, and thus a large number of trainable parameters. As this is not feasible and we want to investigate the performance of hybrid architectures, here we employ a RNN as used by [van den Oord et al., 2019] instead and therefore use a context network consisting of one single LSTM layer with 256 channels to all of the three encoder architectures.

3.4.2 Language Model: Next nucleotide prediction

In order to predict the next nucleotide given a sequence, we adapted the DanQ architecture. Therefore, the final layer was replaced by a dense layer with softmax

activation and four channels, so that a probability for each of the four possible nucleotides is output, and the bidirectional LSTM is replaced by a unidirectional LSTM. To align the previously defined modified CPC architecture, the input sequence has a length of 500 like the patches within the CPC framework.

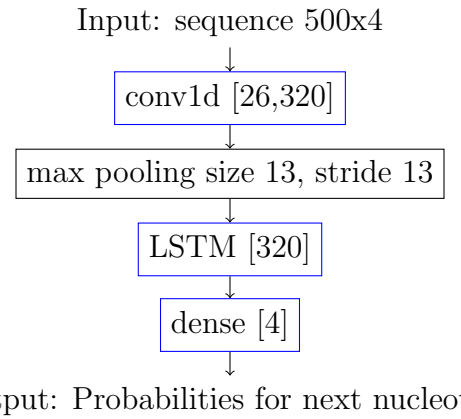

Figure 3.5: Adapted one-dimensional DanQ architecture for next nucleotide prediction. The input is one sequence, on which a one dimensional convolutional layer with kernel size 26 and 320 channels, max pooling, a LSTM layer with 320 channels returning the hidden state output for each input time-step, and finally a dense layer with 4 channels and softmax activation are applied. The output is the prediction of the next nucleotide.

3.5 Downstream Task Evaluation

To determine the quality of the representations, we apply three downstream evaluation methods: (1) Linear classification via regularization using lasso regression, (2) linear classification by adding a single linear layer to the self-supervised network, setting all layers except the added one as untrainable for pure linear evaluation, or (3) linear classification by adding a single linear layer to the self-supervised network keeping the weights trainable and setting a small learning rate for fine-tuning. While the S^4L models are trained in two steps, first on purely unsupervised data and then on supervised data, (4) baseline models are added for comparison. Here, the same architectures as those of the S^4L models are used, but in contrast, these models are trained in one stage on each of the the evaluation tasks, using only labeled data from the beginning and subsequently skipping training on the pre-text task. Since the first two methods are fully linear models, the information obtained by the representation that is relevant for the evaluation task must be linearly separable in the

representation space in order to perform well. The lasso regression is constructed using the `glmnet` [Friedman et al., 2010] with `mlr3` as the general framework [Lang et al., 2019] for evaluation of the Gram staining and the Bacteria-Virus classification tasks. Lasso regression could not be conducted for the chromatin features classification task because there are 919 targets and the model would have to be computed separately and independently for each task, which would require a large amount of computational resources.

Since S^4L models were able to surpass the performance of fully supervised models in other domains even when the amount of labeled data was reduced [Hénaff et al., 2020], we compare training using 0.1%, 1%, 10%, and 100% of the available labeled data for the downstream task evaluation. To assess the quality of the learned representations, they are classified in a supervised manner. As mentioned beforehand, the primary downstream task for evaluation is the classification of bacterial properties, more precisely the classification of Gram-positive and Gram-negative bacteria. Additional downstream tasks include taxonomic Bacteria-Virus classification of bacterial, human, and viral data, as well as classification of chromatin effects in human sequence alterations. The meanings of the targets as well as the origin of the labeled data used are explained in more detail in the following.

3.5.1 Measures

The measures that were evaluated are the log-loss score and balanced accuracy for all tasks, the area under the ROC Curve (AUC) for binary tasks, the F1 score for the chromatin features task, and the macro-averaged-F1 score for the Gram staining and Bacteria-Virus tasks. For the Gram staining task, the macro-averaged-F1 score was applied instead of the F1 score because the latter requires one class to be the one of interest, whereas for this task both classes are equally meaningful. An overview of the measures used per task is shown in Table 3.5.

Table 3.5: Measures that were evaluated for the respective tasks

Task	Measures Applied				
	Log-Loss	Balanced Accuracy	AUC	F1	Macro-Averaged-F1
Gram Staining Classification	✓	✓	✓		✓
Bacteria-Virus Classification	✓	✓			✓
Chromatin Features Classification	✓	✓	✓	✓	

For the true value y_i, the number of observations N and the predicted probability

p, the log-loss score is defined as follows

$$\text{log-loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right]. \tag{3.1}$$

A high log-loss value indicates a less accurate prediction, a low value indicates a more accurate prediction. This value is interpreted dependent on the data distribution by comparing it to the "naive" log-loss that results from making random predictions with the distribution of the classes as probabilities [Vovk, 2015, Dembla, 2021].

The AUC is a measure for binary tasks and compares the probability of the prediction of each class with respect to the true value. A predictor giving an AUC score of 0.5 predictions statistically independent from the outcome, while an AUC score of 1 denotes that all predictions were correct.

Another measure for binary tasks is the F1 score, which compares the probability of predicting each class with respect to the true value as well by computing the harmonic mean of Precision and Recall. Given a confusion matrix

	Predicted 0	Predicted 1
True 0	# True Negatives	# False Positives
True 1	# False Negatives	# True Positives

and

$$Precision = \frac{\#True\ Positives}{\#True\ Positives\ +\ \#False\ Positives} \tag{3.2}$$

$$Recall = Sensitivity = \frac{\#True\ Positives}{\#True\ Positives\ +\ \#False\ Negatives} \tag{3.3}$$

$$Specificity = \frac{\#True\ Negatives}{\#True\ Negatives\ +\ \#False\ Positives} \tag{3.4}$$

F1 is calculated by the following equation:

$$F1 = 2 \times \frac{Precision\ \times\ Recall}{Precision\ +\ Recall} \tag{3.5}$$

A high F1 score results from correct identification of the class of interest, a low F1 score from a high ratio of incorrect predictions. For non-binary classification tasks, the macro-averaged-F1 score is used, a weighted sum of F1 scores computed for each pair of classes. The last measure used is balanced accuracy, which is the proportion of correctly classified data points for multiple groups by averaging the accuracy of each class respectively. The balanced accuracy is calculated by

$$Balanced\ Accuracy = \frac{Sensitivity\ +\ Specificity}{2} \tag{3.6}$$

If there are more than two classes, the balanced accuracy is calculated for each class separately and the sum of the balanced accuracy of all classes is divided by the number of classes. A naive prediction, e.g., a majority prediction, results in a value of $\frac{1}{number\ of\ classes}$, a perfect prediction results in a balanced accuracy of one [Grandini et al., 2020].

Chapter 4

Results

4.1 Experiments

To achieve the highest possible performance, numerous hyperparameters must be set in the beginning of the training of a model. The four most prominent hyperparameters for our models are the learning rate, the embedding scale, the maximum number of samples $n_{\max}$ per FASTA-file, and the model depth. At the beginning of the project, we found that the set learning rate and the embedding scale affect the performance of the model in the first epoch, which again affects the performance in the later course of training. Therefore, we evaluate the achieved validation performance under certain learning rates or embedding scales by a benchmark with all other values fixed. The model architecture used here is the CPC-network with ResNet-50 as the encoder and the training data are genome sequences of bacteria.

We performed three experiments, whereof the first one is the evaluation of different learning rates for choosing a learning rate for the first epoch. In the later epochs, Cosine Annealing is applied with settings based on the previous result. The second experiment is the evaluation of different embedding scales given the best learning rate from the first experiment. The embedding scale remains constant over the training. The third experiment evaluates the different architectures we chose for the encoder within the one-dimensional CPC framework, with respect to different settings for $n_{\max}$, for which the values 16 and 80 are compared, respectively.

4.1.1 Learning rate

In the first experiment, different learning rates were evaluated based on the resulting InfoNCE-loss. We ran the model three times each for the learning rates 4×10^{-5}, 4×10^{-4}, 4×10^{-3}, and 1×10^{-2}. The results are shown in Figure 4.1. The loss resulting from training at learning rate 1×10^{-2} is larger by a factor of up to 10^6 compared to the loss resulting from training with learning rate 4×10^{-4} and 4×10^{-5},

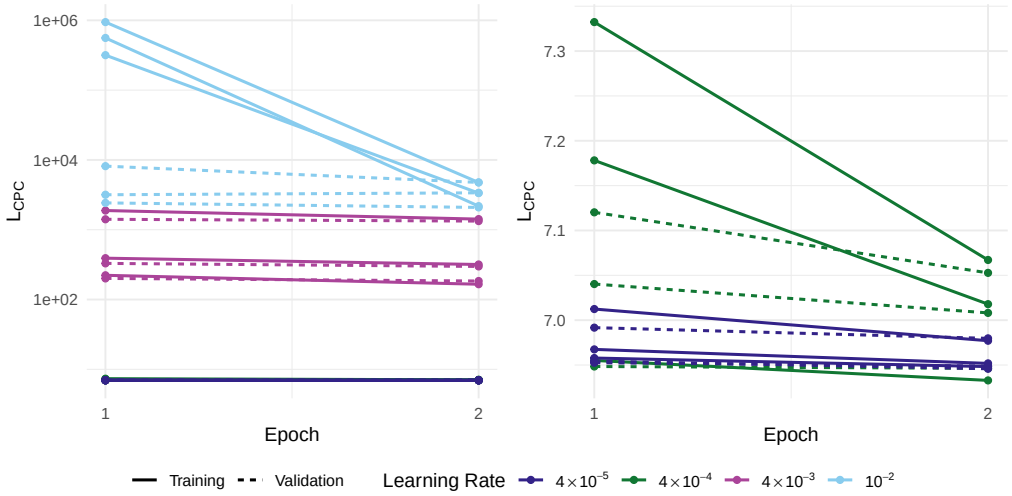

Figure 4.1: Experiment 1: Loss by learning rate, trained for two epochs. Smaller values indicate better performance. The figure on the left shows the results for all learning rates evaluated with a log-scaled y-axis, the figure on the right shows the same data, but zoomed in to the lower part to display a detailed view for the learning rates 4×10^{-5} and 4×10^{-4}. The higher learning rates (4×10^{-3} and $\times 10^{-2}$) cause an initial loss larger by a factor between 10 and 10^4 than the smaller learning rates (4×10^{-5}, 4×10^{-4}).

and the loss resulting from learning rate 4×10^{-3} is larger by a factor of about 10^4 than the loss resulting from training with learning rate 4×10^{-3} and 1×10^{-2}. Therefore, the values $\times 10^{-2}$ and 4×10^{-3} are not chosen as the initial learning rate for the final models. While the two smaller learning rates produce similar results, it is clear from a direct comparison that the learning rate of 4×10^{-5} results in a more stable performance as the difference in losses for the training and validation steps is smaller and for each of the replicates the achieved loss is within a small range of $[6.946;\ 7.012]$, while the range for the learning rate 4×10^{-4} is slightly larger at $[6.933;\ 7.332]$. Consequently, the result of this experiment is to choose the value 4×10^{-5} as the initial learning rate for training the self-supervised models.

4.1.2 Embedding scale

The second experiment compares the resulting InfoNCE-loss during training with different embedding scales. Analogous to the first experiment, we ran the model three times each for the embedding scales 0.001, 0.01, 0.1, 0.5, and 1 with a learning rate of 4×10^{-5}. The results are shown in Figure 4.2. The losses resulting from training with an embedding scale of 1, 0.5, or 0.1 are larger by a factor of up to 1.33 than the losses resulting from training with embedding scale 0.01 and 0.001.

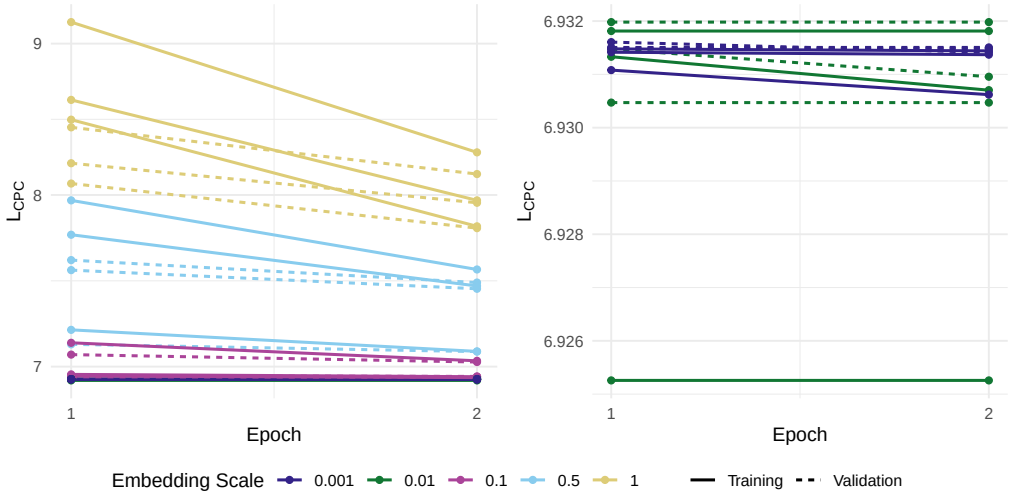

Figure 4.2: Experiment 2: Loss by embedding scale, trained for two epochs. Smaller values indicate better performance. The figure on the left shows the results for all evaluated embedding scales with a log-scaled y-axis, the figure on the right shows the same data, but zoomed into the lower part for displaying a detailed view for the embedding scales 0.001 and 0.01. The higher embedding scales (0.1, 0.5, 1) cause an initial loss larger by the factor of up to 1.33 compared to the smaller embedding scales (0.001, 0.01).

Therefore, the values 1, 0.5, and 4×10^{-3} are not chosen as the initial embedding scale for the final models. While the two smaller embedding scales produce similar results, in direct comparison it is clearly visible that the embedding scale of 0.001 results in a more stable performance. Even though the median test loss is 6.931 for both the embedding scales of 0.01 and 0.001, as visible in Figure 4.2, the training loss differs more from the test loss for the value 0.01. On these grounds, resulting from this experiment, we choose the value of 0.001 as the initial embedding scale for training the self-supervised models.

4.1.3 Encoder architectures and maximum number of samples per FASTA-file

The goal of the third experiment is to find out how certain model architectures, especially with respect to model depth, influence performance. Another aspect is the number of samples $n_{\max}$ taken from a FASTA-file for training a batch. Since some of the sequences, especially those from bacteria, are very long, without setting this argument, it is possible that only one file is considered during one epoch. At the same time, the set $n_{\max}$ affects the resulting InfoNCE loss. For the prediction of a

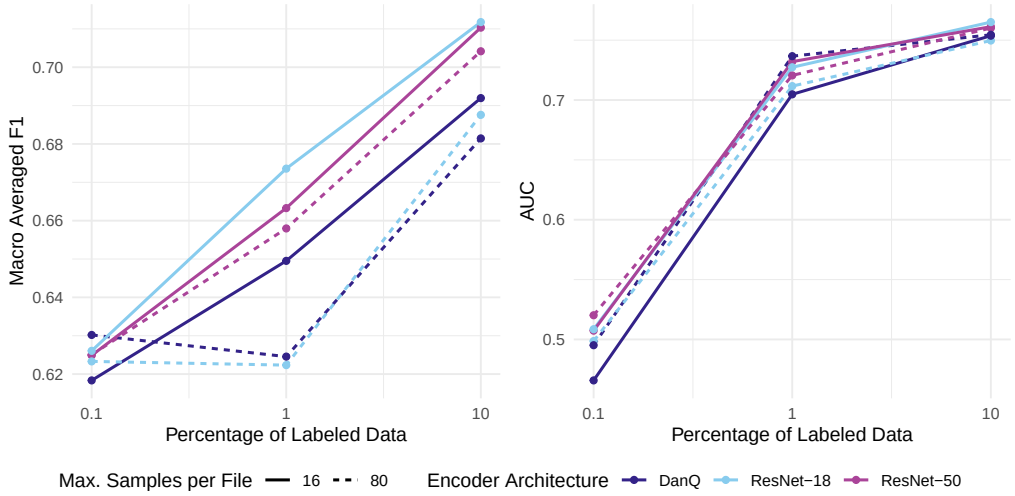

Figure 4.3: Experiment 3: F1 and AUC for Gram staining classification comparing different $n_{\max}$ and encoder architectures. Higher values indicate better performance. Solidly dashed lines display models with $n_{\max} = 16$, dotted lines display models with $n_{\max} = 80$. A High macro-averaged-F1 value (left) as well as high AUC values (right) express better performance. Models with $n_{\max} = 16$ all lead to a higher macro-averaged-F1 compared to the corresponding models with $n_{\max} = 80$, and also to a higher AUC value, except for the model with DanQ encoder.

patch of a sequence at a particular position, the negative samples are other patches from the same sequence, but also other patches from the same batch. Consequently, for batches including only one file, the task is more difficult to solve than for batches including many different files, since distinguishing between different sequences is expected to be more difficult than distinguishing patches within a sequence. The comparison of the InfoNCE losses is therefore not possible between models with different $n_{\max}$. For this experiment, all architectures of the self-supervised model are trained until convergence with $n_{\max} = 16$ and $n_{\max} = 80$ respectively.

The learning rate here is set to 4×10^{-5} and the embedding scale is set to 0.001, as these were the best values resulting from the first two experiments. The self-supervised models are then used to generate representations, and these representations are then put into a lasso regression model that predicts the Gram staining properties. The macro-averaged F1 score as well as the AUC for Gram staining prediction for all models are displayed in Figure 4.3. We observe that in general, models with $n_{\max} = 16$ achieved a higher performance than models with $n_{\max} = 80$, except for the model with the DanQ encoder. Although the performance in terms of AUC is not substantially different for different settings, the model with ResNet-18 encoder and $n_{\max} = 16$ clearly achieves the highest performance regarding macro-averaged F1. Hence, this setting is chosen for the final benchmark to evaluate the

downstream tasks and as a baseline for comparison the BL-CPC-RR18 architecture is chosen and will in the following be refered to as the BL-CPC.

4.2 Evaluation Benchmark

For all S^{4}L models and the baseline model for comparison, the performance on the Gram staining classification task is displayed in Figures 4.4, 4.5, and 4.6, the performance on the Bacteria-Virus classification task is displayed in Figures 4.7, 4.8, and 4.9, and the performance on the chromatin features classification task is displayed in Figures 4.10, 4.11, and 4.12. The dotted lines show the baseline models, the solid lines the S^{4}L models pretrained with the combined pretraining dataset, the dashed lines the S^{4}L models pretrained with bacteria data only, and the different methods applied for evaluation of the respective task are represented by different colors. For simplicity, in the following chapter, the S^{4}L model with CPC architecture with ResNet-18 encoder is referred to as "S^{4}L-CPC", the S^{4}L language model with DanQ architecture applying next nucleotide prediction as a pre-text task is referred to as "S^{4}L-NextNuc", the fully supervised baseline model corresponding to the S^{4}L model with CPC architecture with ResNet-18 encoder is referred to as "BL-CPC", and the fully supervised baseline model corresponding to the language model with DanQ architecture is referred to as "BL-NextNuc". Performance by architecture is compared first, then the respective best models are compared second. Figures including all architectures are included in Appendix B in Figures B.1, B.2, and B.3.

The implementation of the L1 model requires the entire training dataset to be read into working memory at once to compute the representations, and is also not very memory efficient, which when using 100% of the available labeled data results in a working memory usage of at least one terabyte per model. Meanwhile, performing this computation sequentially by splitting requires high temporal resources. Therefore, lasso regression could not be computed for the fully labeled training data, which is why the 100% values for this method are missing.

Due to limitations in computational resources, replicates were only created on a sample basis. The median difference observed is 0.0280 for AUC, 0.0584 for log-loss, 0.0068 for balanced accuracy, and 0.0156 for F1 and macro-averaged-F1. The exact results as well as the models for which replicates are created are depicted in Appendix B, Table B.9

4.2.1 Gram staining classification task

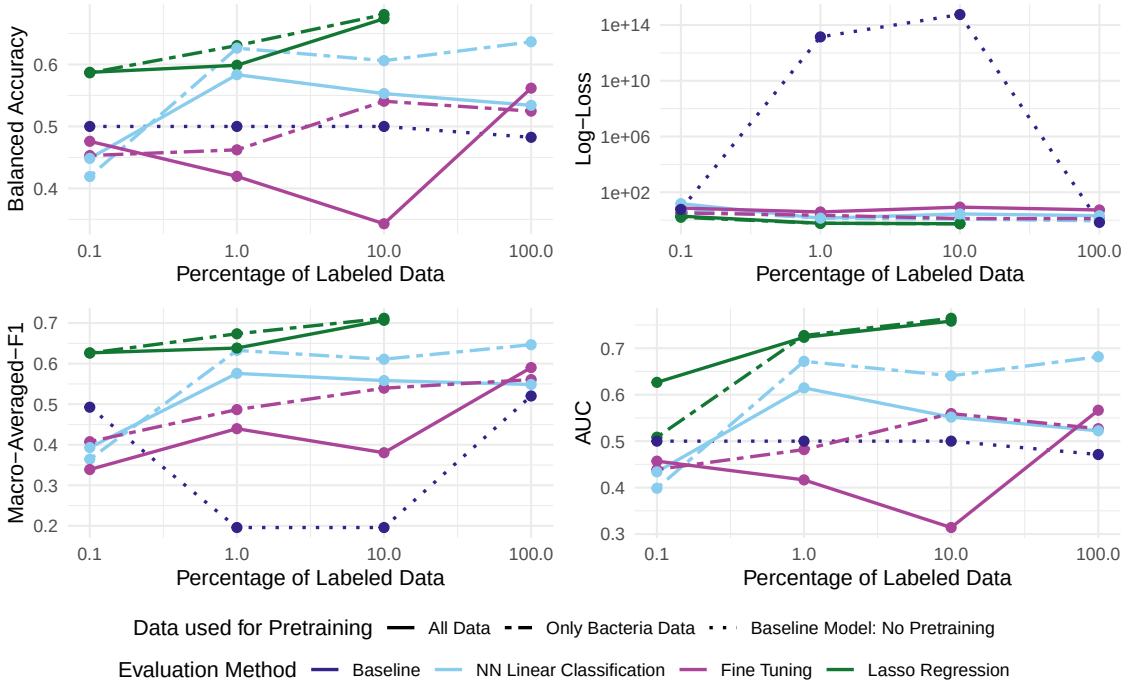

Figure 4.4: Performance on Gram staining classification using CPC architecture with ResNet-18 encoder. For log-loss, smaller values indicate better performance, for Balanced Accuracy, Macro-Averaged-F1, and AUC higher values indicate better performance. The y-axis of the log-loss plot (top right) is log-scaled.

Since the dataset used for training to predict Gram staining properties of a bacterium contains 28% positive and 72% negative samples, a naive prediction in terms of log-loss would lead to a log-loss of

$$-0.4279 \cdot \log(0.4279) - 0.5721 \cdot \log(0.5721) = 0.6827,$$

and a naive prediction in terms of balanced accuracy would lead to a balanced accuracy of

$$\frac{1}{number\ of\ classes} = \frac{1}{2} = 0.5.$$

Because the imbalance within this dataset is factored in by log-loss and AUC to a higher degree compared with the balanced accuracy and F1, although the imbalance is not severe, log-loss and AUC can be considered slightly more informative for this

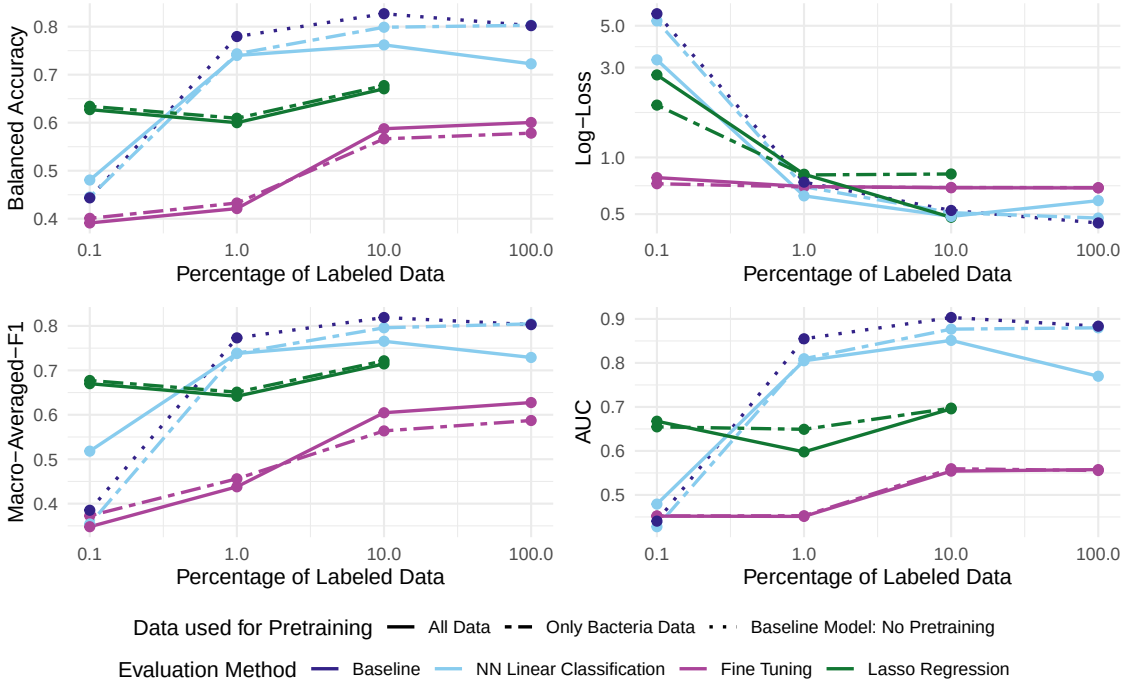

Figure 4.5: Performance on Gram Staining Classification using the language model with DanQ architecture. For log-loss, smaller values indicate better performance, for Balanced Accuracy, Macro-Averaged-F1, and AUC higher values indicate better performance. The y-axis of the log-loss plot (top right) is log-scaled.

task. The detailed values for AUC are listed in Table 4.1, the detailed values for the other measures are listed in Appendix B, Tables B.1, B.2, and B.3.

In Figure 4.4, the performances of S^{4}L-CPC-RN18 and BL-CPC models in the Gram staining classification task are presented. A log-loss better than naive prediction is only achieved by S^{4}L-CPC-RN18 with evaluation by lasso regression trained on 1% and 10% of the available labeled data for both pretraining on the combined pretraining dataset and on bacteria data only, which is subsequently the model that obtains the best performance. BL-CPC and S^{4}L-CPC-RN18 with evaluation by fine-tuning achieved the lowest performance.

Figure 4.5 shows the performance of S^{4}L-NextNuc and BL-NextNuc models on the Gram staining classification task. In contrast to the CPC architecture models, the S^{4}L-NextNuc models that resulted in a log-loss lower than 0.6827 are only achieved by S^{4}L-NextNuc with evaluation by lasso regression trained on 10% of the available labeled data trained on the combined pretraining dataset, by S^{4}L-NextNuc with evaluation by NN linear classification trained on 1%, 10%, and 100% of the available labeled data trained on the combined pretraining dataset, as well as trained

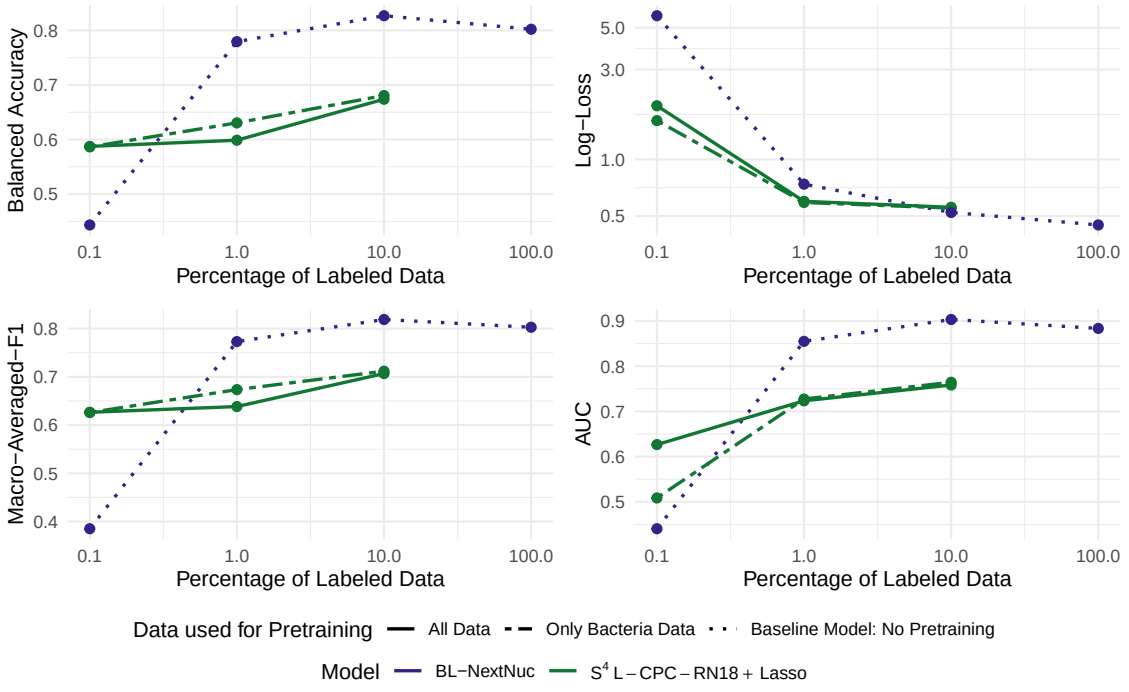

Figure 4.6: Best performances on Gram staining classification. For log-loss, smaller values indicate better performance, for Balanced Accuracy, Macro-Averaged-F1, and AUC higher values indicate better performance. The y-axis of the log-loss plot (top right) is log-scaled.

on 10% and 100% of the available labeled data trained on the bacterial pretraining dataset. Further, BL-NextNuc, trained on 10% and 100% of the available labeled data has resulted in a log-loss lower than 0.6827, and obtains the best performance in total, except when training on 0.1% of the available labeled data, where S^{4}L-NextNuc with evaluation by lasso regression achieves the higher performance. data used for pretraining hardly generates a noticeable impact on performance. S^{4}L-NextNuc with evaluation by fine-tuning achieves the lowest performance.

Finally, the models from both architectures that performed best, S^{4}L-CPC-RN18 with pretraining on only bacteria data with evaluation by lasso regression, S^{4}L-CPC-RN18 with pretraining on the combined pretraining dataset with evaluation by lasso regression, and BL-NextNuc are compared in Figure 4.6. Given the imbalance of the data, we expect the log-loss and AUC values to be more indicative of performance. While the loss for S^{4}L-CPC-RN18 is lower when training on 0.1% or 1% of the data, BL-NextNuc outperforms S^{4}L-CPC-RN18 in terms of log-loss when training on 10% or 100% of the data. The AUC is considerably higher for BL-NextNuc, except for the models trained on 0.1% of the data. The final conclusion for Gram staining

classification is that S^{4}L methods perform better when only a minuscule amount of labeled data is available. Using 0.1% of the labeled dataset measures up to using only two FASTA-files per class, making this observation particularly noteworthy. Simultaneously, in contrast to the results found in recent work as mentioned in section 2.2.1, for this task no definite improvement of performance in comparison with the baseline is confirmed when using more than 0.1% of the data.

Table 4.1: Detailed overview: AUC for Bacteria-Virus classification

% Data	S^{4}L-CPC-RN18							S^{4}L-NextNuc						
	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline
	L1	FT	LC	L1	FT	LC		L1	FT	LC	L1	FT	LC	
0.1	0.509	0.439	0.399	0.627	0.457	0.434	0.500	0.655	0.452	0.428	0.668	0.452	0.479	0.441
1	0.727	0.482	0.672	0.723	0.417	0.615	0.500	0.649	0.453	0.810	0.598	0.451	0.805	0.855
10	0.764	0.559	0.641	0.758	0.314	0.552	0.500	0.697	0.560	0.877	0.696	0.554	0.851	0.903
100		0.527	0.682		0.567	0.522	0.471		0.556	0.880		0.558	0.770	0.884

4.2.2 Bacteria-Virus classification task

For the Bacteria-Virus classification model, a naive prediction in terms of log-loss would result in a log-loss of

$$-0.7447 \cdot \log(0.7447) - 0.2386 \cdot \log(0.2386) - 0.0165 \cdot \log(0.0165) = 0.6291$$

and a naive prediction in terms of balanced accuracy would result in a balanced accuracy of

$$\frac{1}{number\ of\ classes} = \frac{1}{3} \approx 0.3333.$$

The detailed values for balanced accuracy are listed in Table 4.2, the detailed values for the other measures are listed in Appendix B, Tables B.4 and B.5.

Performances obtained by S^{4}L-CPC-RN18 and BL-CPC are displayed in Figure 4.7. A log-loss better than naive prediction is only achieved by S^{4}L-CPC-RN18 with evaluation by lasso regression trained on 1%, 10% and 100% of the available labeled data for both pretraining on the combined pretraining dataset and on bacteria data only, and BL-CPC-RN18 trained on 1% and 100% of the available labeled data. The setting that leads to the best performance here is S^{4}L-CPC-RN18 with lasso regression, achieving the lowest log-loss and the highest macro-averaged-F1 as well as balanced accuracy. Since lasso regression could not be computed for the training with 100% of the data, the best performance here was achieved by BL-CPC.

Figure 4.8 depicts the performance attained by S^{4}L-NextNuc and BL-NextNuc on the Bacteria-Virus classification task. In contrast to the CPC architecture models, half of the S^{4}L-NextNuc models resulted in a log-loss lower than 0.6291, while

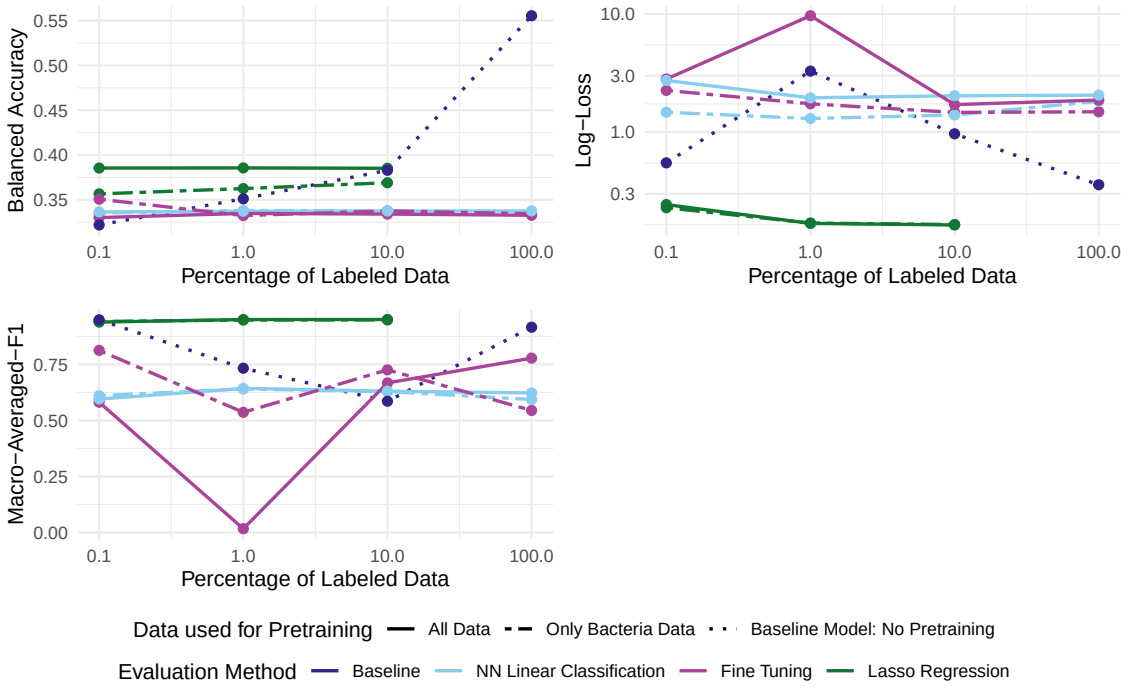

Figure 4.7: Performance on Bacteria-Virus classification CPC architecture with ResNet-18 encoder. For log-loss, smaller values indicate better performance, for Balanced Accuracy and Macro-Averaged-F1 higher values indicate better performance. The y-axis of the log-loss plot (top right) is log-scaled.

we observe that these resulting in a log-loss higher than 0.6291 are the models with evaluation by fine-tuning. While S^{4}L-NextNuc pretrained on the combined pretraining dataset with evaluation by NN linear classification shows unstable performance, as it achieves good results on all points except training using 10% of the labeled data, evaluation by fine-tuning again shows consistently poor performance. Good performances are obtained by S^{4}L-NextNuc with lasso regression in terms of macro-averaged-F1, and S^{4}L-NextNuc with NN linear classification in terms of log-loss and balanced accuracy, while the baseline model achieved a slightly higher value for the latter when training on 100% of labeled data. Since the result with the best trade-off between stability and good performance measure results is obtained by S^{4}L-NextNuc with evaluation by NN linear classification, this model is further compared with the best model with CPC architecture, the S^{4}L-CPC-RN18 with evaluation by lasso regression.

Finally, the models of both architectures that achieved the best performances are compared: S^{4}L-CPC-RN18 with evaluation by lasso regression with pretraining on only bacteria data and S^{4}L-NextNuc with pretraining both data sets with evaluation

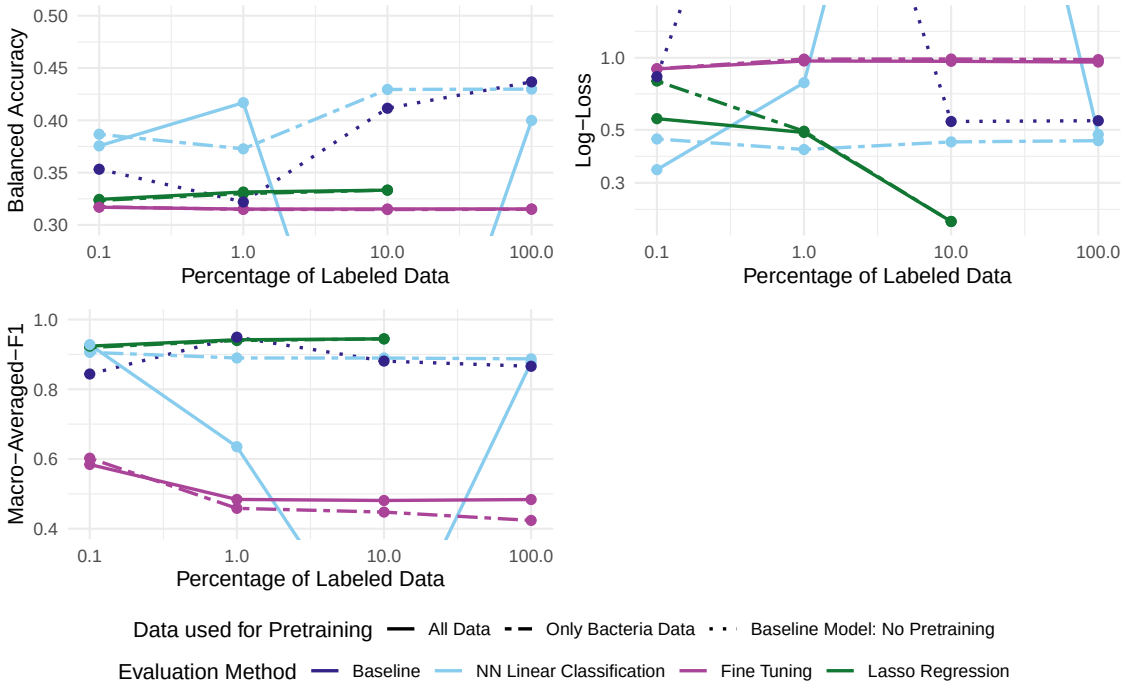

Figure 4.8: Performance on Bacteria-Virus classification using the language model with DanQ architecture. For enhanced visibility of individual performances, the plots were scaled to exclude outliers; full plots are included in Appendix B, Figure B.2. For log-loss, smaller values indicate better performance, for Balanced Accuracy and Macro-Averaged-F1 higher values indicate better performance. The y-axis of the log-loss plot (top right) is log-scaled.

by NN linear classification. The resulting values are shown in Figure 4.9. While the loss is lower and the macro-averaged-F1 is higher for S^{4}L-NextNuc, S^{4}L-CPC-RN18 outperforms S^{4}L-NextNuc in terms of balanced accuracy when 0.1% or 1% of the labeled data is used for training. Contrary to the Gram staining classification task, S^{4}L methods clearly outperform fully supervised methods in this task.

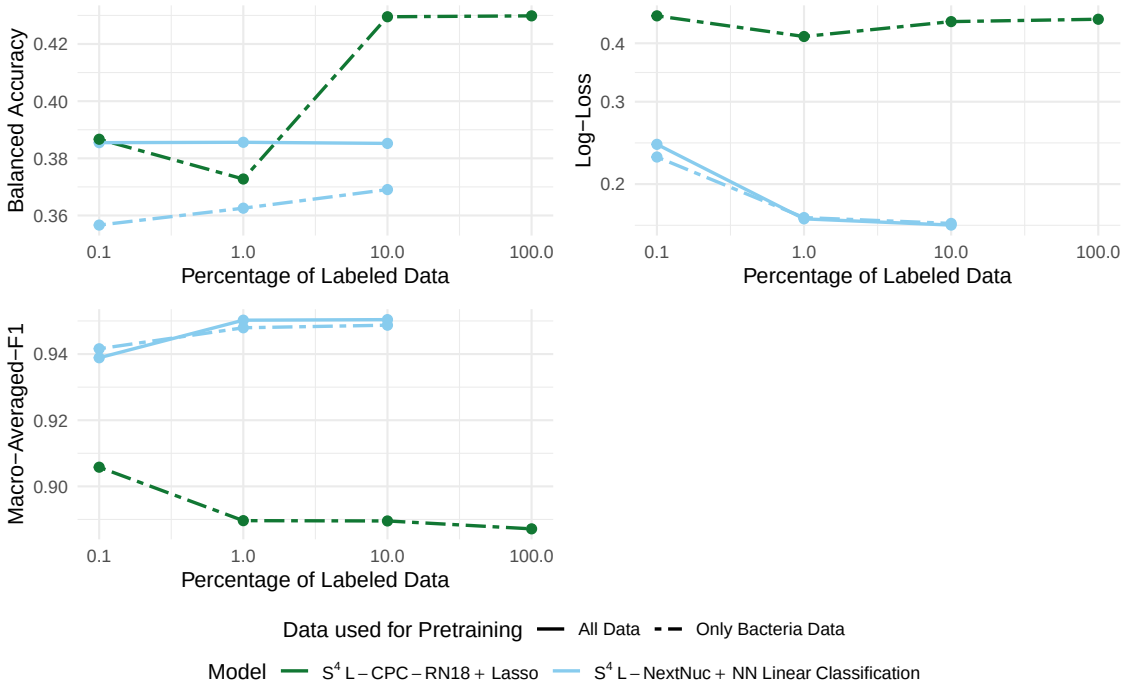

Figure 4.9: Best performances on Bacteria-Virus classification. For log-loss, smaller values indicate better performance, for Balanced Accuracy and Macro-Averaged-F1 higher values indicate better performance. The y-axis of the log-loss plot (top right) is log-scaled.

Table 4.2: Detailed overview: Balanced accuracy for Bacteria-Virus classification

% Data	S^{4}L-CPC-RN18							S^{4}L-NextNuc						
	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Base-line	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Base-line
	L1	FT	LC	L1	FT	LC		L1	FT	LC	L1	FT	LC	
0.1	0.357	0.351	0.335	0.385	0.330	0.336	0.322	0.323	0.317	0.387	0.324	0.317	0.376	0.353
1.0	0.363	0.332	0.338	0.386	0.335	0.337	0.351	0.330	0.315	0.373	0.331	0.315	0.417	0.322
10.0	0.369	0.338	0.338	0.385	0.334	0.337	0.383	0.333	0.315	0.430	0.333	0.315	0.005	0.412
100.0		0.334	0.337		0.333	0.338	0.555		0.315	0.430		0.315	0.400	0.437

4.2.3 Chromatin features classification task

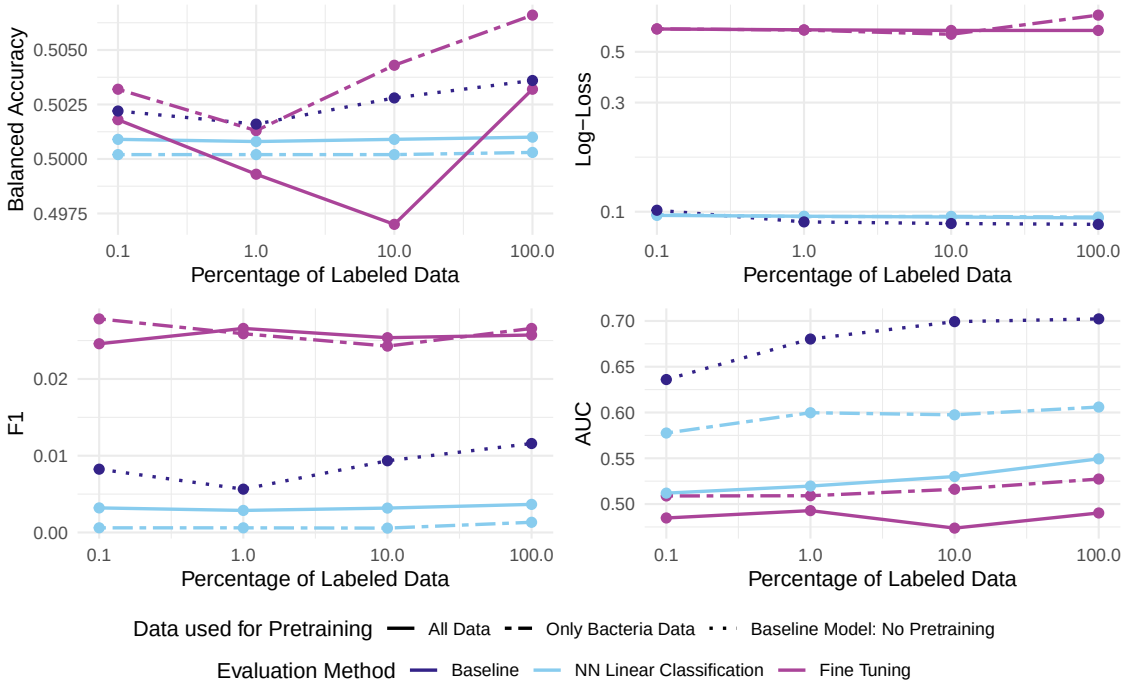

Figure 4.10: Performance on chromatin features classification using the CPC architecture with ResNet-18 encoder. For log-loss, smaller values indicate better performance, for Balanced Accuracy, F1, and AUC higher values indicate better performance. The y-axis of the log-loss plot (top right) is log-scaled.

The distribution of the data used for training to classify different chromatin features is highly unbalanced for all targets, with on average only 2% of the observations being positive. Because this imbalance is factored in by log-loss and AUC to a greater extent compared with the balanced accuracy and F1, these measures can be regarded more meaningful for this task. The detailed values for AUC are listed in Table 4.3, the detailed values for the other measures are listed in Appendix B, Tables B.6, B.7, and B.8. An example for the interpretability issues caused by the imbalance is illustrated in Figure 4.10, which visualizes the performance of S^{4}L-CPC-RN18 and BL-CPC. S^{4}L-CPC-RN18 with fine-tuning evaluation achieved the best performance regarding balanced accuracy and F1, while achieving the worst performance in terms of log-loss and AUC, since majority prediction does not induce a reduction in absolute values for F1 and balanced accuracy. Thus, the best performance here is achieved by BL-CPC, yielding a constant performance, as the values for all AUC, balanced accuracy, and F1 are high and for log-loss, they are

low. If d_i is the percentage of positive observations in target i, naive prediction results in a log-loss of

$$\frac{1}{919}\sum_{i=1}^{919} -d_i \log d_i - (1-d_i)\log(1-d_i) = 0.0956.$$

A log-loss better than naive prediction is only achieved by S^{4}L-CPC-RN18 with evaluation by NN linear classification trained on 1%, 10% and 100% of the available labeled data for both pretraining on the combined pretraining dataset and on bacteria data only and BL-CPC-RN18 trained on 1%, 10% and 100% of the available labeled data.

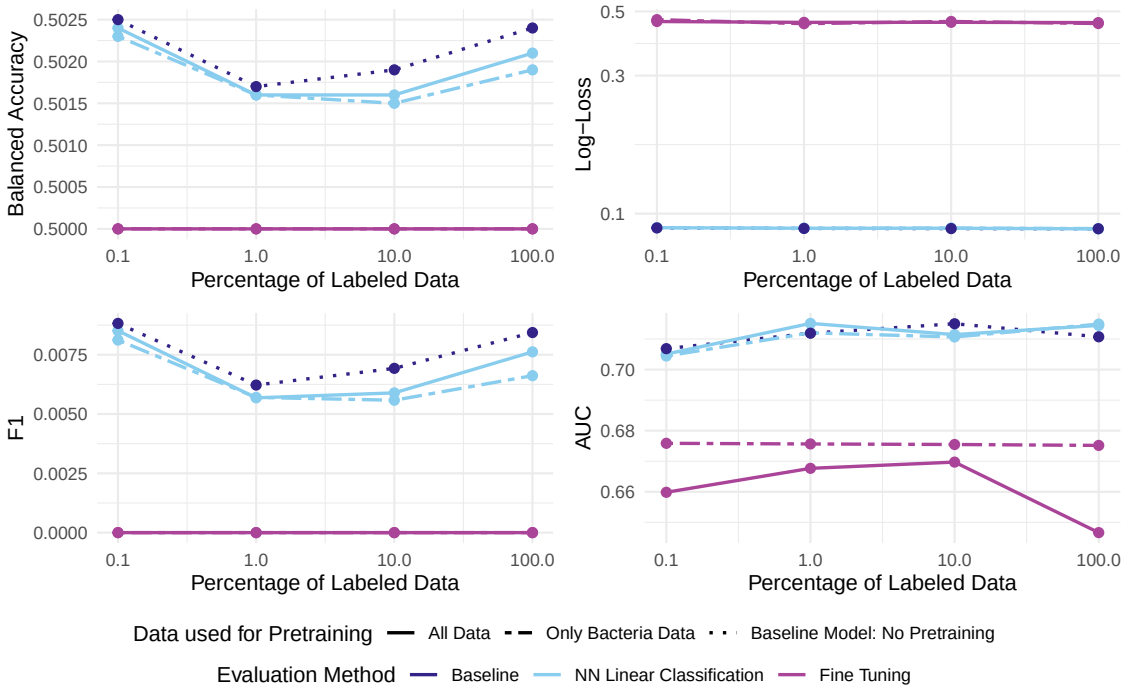

Figure 4.11: Performance on chromatin features classification using the language model with DanQ architecture. For log-loss, smaller values indicate better performance, for Balanced Accuracy, F1, and AUC higher values indicate better performance. The y-axis of the log-loss plot (top right) is log-scaled.

Furthermore, for S^{4}L-NextNuc and BL-NextNuc, whose performances are shown in Figure 4.11, S^{4}L-NextNuc with evaluation by fine-tuning achieved the worst, and BL-NextNuc the best results, whereas S^{4}L-NextNuc with evaluation by NN-classification is able to achieve a performance with similarly high results. A log-loss

better than naive prediction is again achieved by S^{4}L-NextNuc with evaluation by NN linear classification with all pretraining settings and BL-CPC-RN18 also with all pretraining settings. Overall, no model was able to obtain high performance

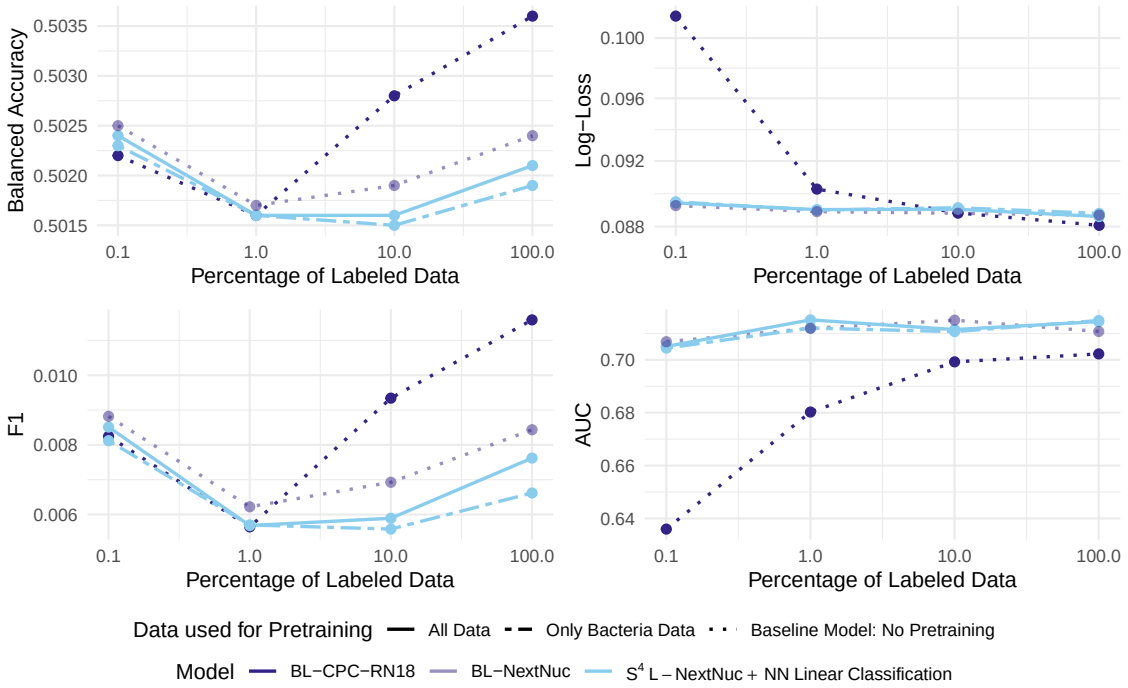

Figure 4.12: Best performances on chromatin features classification. For log-loss, smaller values indicate better performance, for Balanced Accuracy, F1, and AUC higher values indicate better performance. The y-axis of the log-loss plot (top right) is log-scaled.

in comparison with methods examined in recent work. While the original DanQ architecture by Quang and Xie [2016] and the DeepSEA architecture by Zhou and Troyanskaya [2015] achieved an AUC of > 0.85, and the S^{4}L "Self-GenomeNet" architecture introduced by Anonymous [2022] achieved an AUC of > 0.75, all of the evaluated methods in this thesis have only achieved an AUC of up to 0.715, which is comparable to the fully supervised baseline methods presented by Anonymous [2022]. Because no log-loss, F1, or balanced accuracy was published in the previously mentioned studies, we cannot compare these measures with the values that resulted from our models. Consequently, the S^{4}L models we evaluated for this task were not able to outperform fully supervised methods or the Self-GenomeNet architecture including self-supervision.

Table 4.3: Detailed overview: AUC for Chromatin features classification

% Data	S^{4}L-CPC-RN18					S^{4}L-NextNuc				
	Pretrained on Bacteria Dataset		Pretrained on Combined Dataset		Baseline	Pretrained on Bacteria Dataset		Pretrained on Combined Dataset		Baseline
	FT	LC	FT	LC		FT	LC	FT	LC	
0.1	0.509	0.578	0.485	0.512	0.636	0.676	0.704	0.660	0.705	0.707
1	0.509	0.600	0.493	0.520	0.680	0.676	0.712	0.668	0.715	0.712
10	0.516	0.597	0.474	0.530	0.699	0.675	0.711	0.670	0.711	0.715
100	0.527	0.606	0.490	0.549	0.702	0.675	0.715	0.647	0.714	0.711

4.3 Interpretability of the Representations

To investigate the information contained in the fitted representations, we created T-distributed stochastic neighbor embedding (TSNE) plots. This method reduces the information of a representation to a two-dimensional vector based on stochastic neighbor embedding [Hinton and Roweis, 2002], which can be visualized by interpreting this vector as a location in a two-dimensional space. The TSNE plots created are depicted in Figure 4.13 and 4.14. As the targets are based on different datasets, the values in these two figures are not comparable.

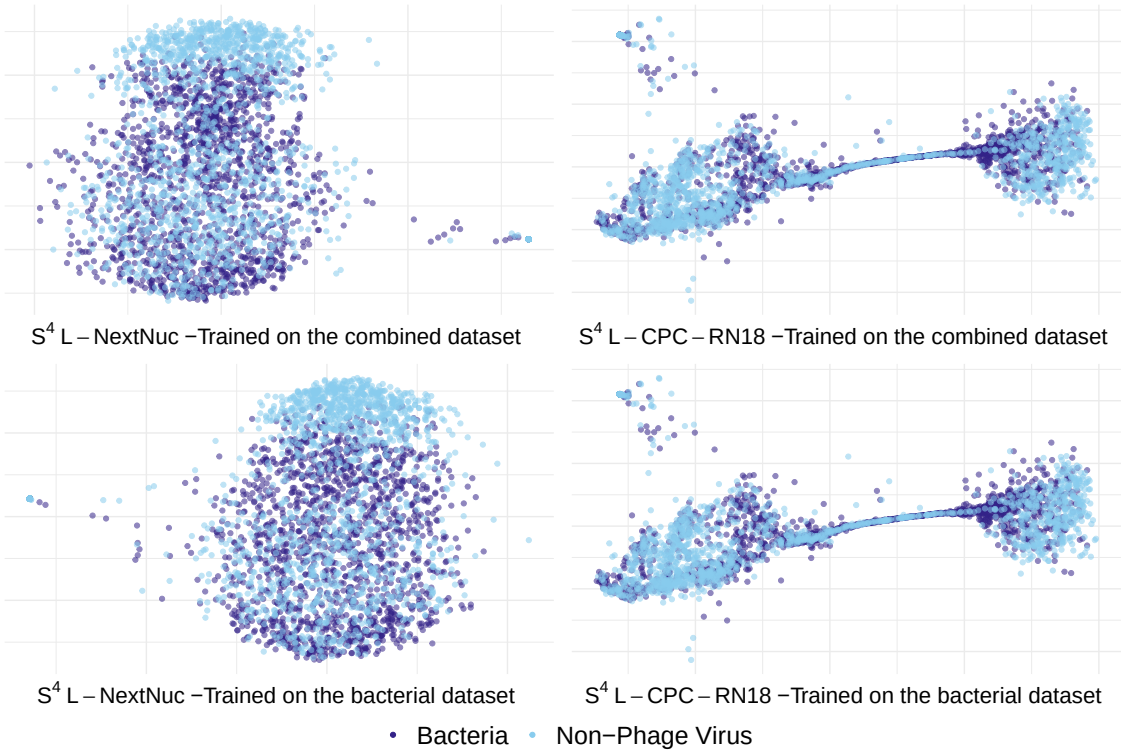

Figure 4.13: T-distributed stochastic neighbor embedding for Gram staining target.

Generally, clustering of Gram-staining properties based on the representations is somewhat present as depicted in Figure 4.13, visualizing the representations produced by the respective S^{4}L models simplified by positions on a graph and different classes represented by color marking. In contrast to the CPC model with ResNet-18 encoder, the language model possesses an enhanced ability to sort the Gram-positive bacteria into a clearly recognizable non-exhaustive cluster. Partial clustering of the Gram-positive bacteria is present in all models, but a full separation of the Gram-staining property cannot be produced by the trained models.

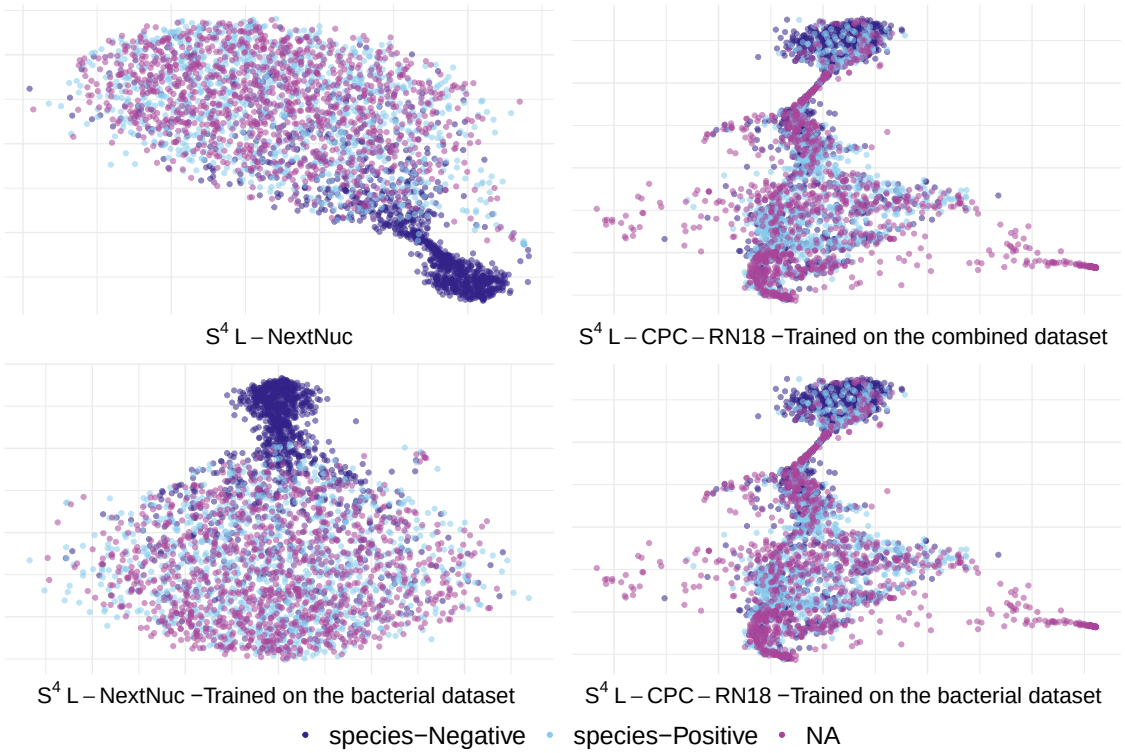

Figure 4.14: T-distributed stochastic neighbor embedding for Bacteria-Virus target.

As further observed in the visualization of the representation with respect to the Gram staining labels, the model is also capable of distinguishing bacteria, phage-viruses, and non-phage-viruses, which the TSNE-plot in Figure 4.14 depicts. Again, it is clearly visible that the language model recognizes boundaries more distinctly than the CPC model. Especially the group of bacteria is strongly distinguished from phage viruses and non-phage viruses, which can be explained by a higher genetic similarity between different virus types compared to the similarity between viruses and bacteria. The CPC model also appears to capture clusters, but here the boundaries are more ambiguously separated. However, a clear clustering of bacteria can be seen here as well.

Chapter 5

Discussion and Conclusion

5.1 Future Aspects

One substantial issue we faced while doing experiments for this work was the enormous computational overhead involved in the analyses. For the pretraining of a self-supervised model, the training took between one and three weeks to converge, even when using large GPU machines. The semi-supervised step, using the trained self-supervised model as a baseline, required up to two more weeks to converge. As described in Chapter 3, we had to constrain some parameters to specific values due to computational and temporal limitations. For some of the methods, such as the lasso models, the execution was not feasible at all. The analyses performed took multiple GPU years already.

On top of this, we did not have the opportunity to create statistical replicates for all models. Instead, replicates were created on a sample basis, and the replicates created also showed a slight instability of the models in terms of the resulting loss. This raises the question of whether interpretation of the results presented is possible at this point, or whether multiple replications are necessary before an interpretation can be more reasonable, which is the expectation in this case.

While many more experiments were already implemented and prepared, the resources were not sufficient to run all of these. The implemented experiments include additional settings for downstream task evaluation, such as using not only one linear layer on top of the pretrained network, but also different layer types and additional subsequent layers, as described as "efficient classification" by Hénaff et al. [2020], the application of a "PixelCNN" [van den Oord et al., 2016] or a higher-dimensional LSTM as context network. The latter was not performed due to an emerging high number of trainable parameter causing memory errors.

As mentioned beforehand, the lasso regression is missing for the chromatin features classification task. For this purpose, instead of the `mlr3` framework, an additional framework using the `biglasso` package [Zeng and Breheny, 2017] was already

prepared, which provides a built-in option for paralellization of the training, advantageous for training all of the 919 targets. Thus, another possible avenue is the computation of the lasso regression to assess its performance in comparison to the already computed evaluation methods. Additionally, the missing lasso regression models for Gram staining and Bacteria-Virus classification using 100% of the available training data can be computed with more computational resources. Other interesting experiments for future work include pretraining on only human genome sequence data for the chromatin features classification task, as the labeled data contains only human genome sequences, and evaluating supplementary targets such as the CRISPR locus within a sequence and protein binding affinity.

5.2 Conclusion

In general, it can be concluded that the language model S^4L-NextNuc is better suited as a self-supervised baseline and performs better on the downstream evaluation tasks than the self-supervised CPC model, which is also confirmed by the TSNE-plots. Due to the complexity of the S^4L-CPC-architectures, the higher performance by S^4L-NextNuc is against the expectations. While the S^4L models were able to outperform the fully-supervised baseline models on the Bacteria-Virus classification task and on the Gram staining classification task when using only 0.1% of the labeled data for training, the results are otherwise worse in comparison with the fully supervised models. The respresentations created by self-supervised pretraining are useful for aiding the distinction of different classes while being in need of improvements. Another discovery was the low performance of almost all models applying fine tuning for evaluation, which we are unable to explain at this point without further experiments.

In addition, we found that the data used for pretraining, i.e., whether only bacterial data or the combined human, viral, and bacterial data were used, did not have a striking effect on the performance in the downstream tasks and therefore higher diversity within the training data was not proven to lead to the extraction of higher-level slow-features. However, the resemblance of the performances of the same model trained on only bacterial data or the combined data indicate stability of the performances. As no promising results were obtained for the prediction of chromatin features, no conclusion on the suitability of the representations for transfer learning tasks can be drawn.

While the findings in recent work demonstrated an acceleration of performance by using a semi-supervised model in combination with self-supervision by befitting from the quantity of unlabeled genome sequences, we are unable to find any significant improvements by S^4L models. Albeit Anonymous [2022] showed a boost in performance from S^4L models. In the preliminary experiment described in Chapter 4.1.3,

we have compared different architectures, of which one, the S^4L-CPC-DanQ resembles the architecture applied by Anonymous [2022]. The main difference between our and their technique is the design of the pre-text task and the length of the input sequence, as their input sequence consists of 150 nucleotides, while ours consists of 6700 nucleotides. A benchmark which directly compares the methods by evaluating the influence of each different setting could provide a better insight into the reasons for the difference in performance.

Although the decision was made in favor of the deep architectures using ResNets because of the promising results from Hénaff et al. [2020], it is possible that what they found in the image domain does not apply to genome sequences. Possible advantages using shallower networks are also evidenced by the better performance of the language model in comparison to the CPC model. The final conclusion we draw with reservations is that the deep networks applied for self-supervision are not highly beneficial for downstream task evaluation for Gram staining classification and chromatin features classification, while slightly outperforming the fully supervised baseline models for Bacteria-Virus classification. Though, to enable reliable judgments, further analyses given high computational and temporal resources are needed.

List of Figures

List of Tables

Bibliography

D. D. F. Adiwardana, A. Matsukawa, and J. Whang. Using generative models for semi-supervised learning. In *Medical image computing and computer-assisted intervention–MICCAI*, volume 2016, pages 106–14, 2016.

M. Akhtar, N. Mohsin, A. Zahak, M. R. Ain, P. Pillai, P. Kapur, and M. Z. Ahmad. Antimicrobial sensitivity pattern of bacterial pathogens in urinary tract infections in south delhi, india. *Reviews on recent clinical trials*, 9, 11 2014. doi: 10.2174/1574887109666141127104220.

B. Alberts. The cell as a collection of protein machines: Preparing the next generation of molecular biologists. *Cell*, 92(3):291–294, 1998. doi: 10.1016/s0092-8674(00)80922-8.

B. Alipanahi, A. Delong, M. Weirauch, and B. Frey. Predicting the sequence specificities of dna- and rna-binding proteins by deep learning. *Nature biotechnology*, 33, 07 2015. doi: 10.1038/nbt.3300.

Anonymous. Self-genomenet: Self-supervised learning with reverse-complement context prediction for nucleotide-level genomics data. In *Submitted to The Tenth International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=92awwjGxIZI`. under review.

E. Asgari and M. R. K. Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *Plos One*, 10(11), 2015. doi: 10.1371/journal.pone.0141287.

B. Atal and M. Schroeder. Predictive coding of speech signals and subjective error criteria. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 3, pages 573 – 576, 05 1978. doi: 10.1109/ICASSP.1978.1170564.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives, 2014.

N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription, 2012.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

C. A. Chapelle. English language learning and technology. *Language Learning & Language Teaching*, 2003. doi: 10.1075/lllt.7.

O. Chapelle, S. Bernhard, and A. Zien. *Semi-supervised learning*. The MIT Press, 2006.

H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke. The rise of deep learning in drug discovery. *Drug Discovery Today*, 23(6):1241–1250, 2018. ISSN 1359-6446. doi: https://doi.org/10.1016/j.drudis.2018.01.039. URL `https://www.sciencedirect.com/science/article/pii/S1359644617303598`.

T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020. URL `https://arxiv.org/abs/2002.05709`.

F. Chollet, J. Allaire, et al. R interface to keras. `https://github.com/rstudio/keras`, 2017.

G. Dembla. Intuition behind log-loss score, Dec 2021. URL `https://towardsdatascience.com/intuition-behind-log-loss-score-4e0c9979680a`.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction, 2016.

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL `https://www.jstatsoft.org/v33/i01/`.

K. Friston. A theory of cortical responses. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 360:815–36, 05 2005. doi: 10.1098/rstb.2005.1622.

F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, 1999.

S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations, 2018.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.

M. Grandini, E. Bagli, and G. Visani. Metrics for multi-class classification: an overview, 2020.

M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `https://proceedings.mlr.press/v9/gutmann10a.html`.

J. Harrow, A. Nagy, A. Reymond, T. Alioto, L. Patthy, S. E. Antonarakis, and R. Guigó. Identifying protein-coding genes in genomic sequences. *Genome Biology*, 10(1):201, Jan 2009. doi: 10.1186/gb-2009-10-1-201.

H. R. Hassanzadeh and M. D. Wang. Deeperbind: Enhancing prediction of sequence specificities of DNA binding proteins. *CoRR*, abs/1611.05777, 2016. URL `http://arxiv.org/abs/1611.05777`.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

G. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *NIPS*, volume 15, pages 833–840. Citeseer, 2002.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. ISSN 0027-8424. doi: 10.1073/pnas.79.8.2554. URL `https://www.pnas.org/content/79/8/2554`.

G. J. Hucker and H. J. Conn. Methods of gram staining. *Technical Bulletin No.93*, 1923.

O. J. Hénaff, A. Srinivas, J. D. Fauw, A. Razavi, C. Doersch, S. M. A. Eslami, and A. van den Oord. Data-efficient image recognition with contrastive predictive coding, 2020.

A. Jan, Z. Hasan, H. Shah, R. Ullah, I. Ahmad, and M. Younas. An investigation of the bacterial flora causing spoilage of fishes at board fish market, peshawar, pakistan. *Pakistan journal of zoology*, 46:1371–1375, 10 2014.

K. G. Joensen, F. Scheutz, O. Lund, H. Hasman, R. S. Kaas, E. M. Nielsen, and F. M. Aarestrup. Real-time whole-genome sequencing for routine typing, surveillance, and outbreak detection of verotoxigenic escherichia coli. *Journal of Clinical Microbiology*, 52(5):1501–1510, Feb 2014. doi: 10.1128/jcm.03617-13.

K. Kawakami, L. Wang, C. Dyer, P. Blunsom, and A. van den Oord. Unsupervised learning of efficient and robust speech representations, 2020. URL `https://openreview.net/forum?id=HJe-blSYvH`.

D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling. Semi-supervised learning with deep generative models. *CoRR*, abs/1406.5298, 2014. URL `http://arxiv.org/abs/1406.5298`.

A. Kolesnikov, X. Zhai, and L. Beyer. Revisiting self-supervised visual representation learning, 2019.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, dec 2019. doi: 10.21105/joss.01903. URL `https://joss.theoj.org/papers/10.21105/joss.01903`.

X. H. Le, H. Ho, G. Lee, and S. Jung. Application of long short-term memory (lstm) neural network for flood forecasting. *Water*, 11:1387, 07 2019. doi: 10.3390/w11071387.

Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995.

I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

A. X. Lu, H. Zhang, M. Ghassemi, and A. Moses. Self-supervised contrastive learning of protein representations by mutual information maximization. *bioRxiv*, 2020. doi: 10.1101/2020.09.04.283929. URL `https://www.biorxiv.org/content/early/2020/11/10/2020.09.04.283929`.

E. A. Miao and S. I. Miller. Bacteriophages in the evolution of pathogen-host interactions. *Proceedings of the National Academy of Sciences*, 96(17):9452–9454, 1999. doi: 10.1073/pnas.96.17.9452.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013. URL `http://arxiv.org/abs/1310.4546`.

R. Mreches, A. C. McHardy, B. Bischl, J. Moosbauer, H. A. Gündüz, S. Klawitter, Z.-L. Deng, E. Franzosa, C. Huttenhower, G. Robertson, E. Asgari, X.-Y. To, M. Binder, and P. C. Münch. Genomenet/deepg: Deepg pre-release version, Oct. 2021. URL `https://doi.org/10.5281/zenodo.5561229`.

K. A. Neilson, N. A. Ali, S. Muralidharan, M. Mirzaei, M. Mariani, G. Assadourian, A. Lee, S. C. V. Sluyter, and P. A. Haynes. Less label, more free: Approaches in label-free quantitative mass spectrometry. *Proteomics*, 11(4):535–553, 2011. doi: 10.1002/pmic.201000553.

A. Odena. Semi-supervised learning with generative adversarial networks, 2016.

D. Quang and X. Xie. DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Research*, 44(11):e107–e107, 04 2016. ISSN 0305-1048. doi: 10.1093/nar/gkw226. URL `https://doi.org/10.1093/nar/gkw226`.

A. Rasmus, H. Valpola, M. Honkala, M. Berglund, and T. Raiko. Semi-supervised learning with ladder network. *CoRR*, abs/1507.02672, 2015. URL `http://arxiv.org/abs/1507.02672`.

L. C. Reimer, A. Vetcininova, J. S. Carbasse, C. Söhngen, D. Gleim, C. Ebeling, and J. Overmann. BacDive in 2019: bacterial phenotypic data for High-throughput biodiversity analysis. *Nucleic Acids Research*, 47(D1):D631–D636, 09 2018. ISSN 0305-1048. doi: 10.1093/nar/gky879. URL `https://doi.org/10.1093/nar/gky879`.

A. S. Rifaioglu, H. Atas, M. J. Martin, R. Cetin-Atalay, V. Atalay, and T. Doğan. Recent applications of deep learning and machine intelligence on in silico drug discovery: methods, tools and databases. *Briefings in Bioinformatics*, 20(5):1878–1912, 07 2018. ISSN 1477-4054. doi: 10.1093/bib/bby061. URL `https://doi.org/10.1093/bib/bby061`.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

E. W. Sayers, M. Cavanaugh, K. Clark, K. D. Pruitt, C. L. Schoch, S. T. Sherry, and I. Karsch-Mizrachi. GenBank. *Nucleic Acids Research*, 49(D1):D92–D96, 11 2020. ISSN 0305-1048. doi: 10.1093/nar/gkaa1023. URL `https://doi.org/10.1093/nar/gkaa1023`.

P. Sermanet, C. Lynch, J. Hsu, and S. Levine. Time-contrastive networks: Self-supervised learning from multi-view observation. *CoRR*, abs/1704.06888, 2017. URL `http://arxiv.org/abs/1704.06888`.

O. Sizar and C. G. Unakal. Gram positive bacteria. *StatPearls [Internet]*, 2020.

L. N. Smith. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2015. URL `http://arxiv.org/abs/1506.01186`.

C. Szalai, F. Oberfrank, E. Pap, K. Szabó-Taylor, A. Falus, S. Tóth, and V. László. *Medizinische Genetik und Genomik*. Typotex, 06 2020. ISBN 978 963 279 188 3.

M. Szummer and T. Jaakkola. Partially labeled classification with markov random walks. In *Advances in Neural Information Processing Systems*, pages 945–952. MIT Press, 2002.

C. Söhngen, A. Podstawka, B. Bunk, D. Gleim, A. Vetcininova, L. C. Reimer, C. Ebeling, C. Pendarovski, and J. Overmann. BacDive – The Bacterial Diversity Metadatabase in 2016. *Nucleic Acids Research*, 44(D1):D581–D585, 09 2015. ISSN 0305-1048. doi: 10.1093/nar/gkv983. URL `https://doi.org/10.1093/nar/gkv983`.

B. Tang, Z. Pan, K. Yin, and A. Khateeb. Recent advances of deep learning in bioinformatics and computational biology. *Frontiers in Genetics*, 10:214, 03 2019. doi: 10.3389/fgene.2019.00214.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the royal statistical society series b-methodological*, 58:267–288, 1996.

A. Trabelsi, M. Chaabane, and A. Ben-Hur. Comprehensive evaluation of deep learning architectures for prediction of DNA/RNA sequence binding specificities. *Bioinformatics*, 35(14):i269–i277, 07 2019. doi: 10.1093/bioinformatics/btz339. URL `https://doi.org/10.1093/bioinformatics/btz339`.

A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016. URL `http://arxiv.org/abs/1606.05328`.

A. van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding, 2019.

V. Vovk. *The Fundamental Nature of the Log Loss Function*, pages 307–318. Springer International Publishing, Cham, 2015. ISBN 978-3-319-23534-9. doi: 10.1007/978-3-319-23534-9_20. URL `https://doi.org/10.1007/978-3-319-23534-9_20`.

R. Wang, T. Zang, and Y. Wang. Human mitochondrial genome compression using machine learning techniques. *Human Genomics*, 13:49, 10 2019. doi: 10.1186/s40246-019-0225-3.

S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL `http://arxiv.org/abs/1605.07146`.

Y. Zeng and P. Breheny. The biglasso package: A memory- and computation-efficient solver for lasso model fitting with big data in r. *ArXiv e-prints*, 2017. URL `https://arxiv.org/abs/1701.05936`.

X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer. S4l: Self-supervised semi-supervised learning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. doi: 10.1109/iccv.2019.00156.

Y. Zhang, J. Yan, S. Chen, M. Gong, D. Gao, M. Zhu, and W. Gan. Review of the applications of deep learning in bioinformatics. *Current Bioinformatics*, 15(8):898–911, 2021. doi: 10.2174/1574893615999200711165743.

J. Zhou and O. Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12, 08 2015. doi: 10.1038/nmeth.3547.

X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

Appendix A

Model Architecture

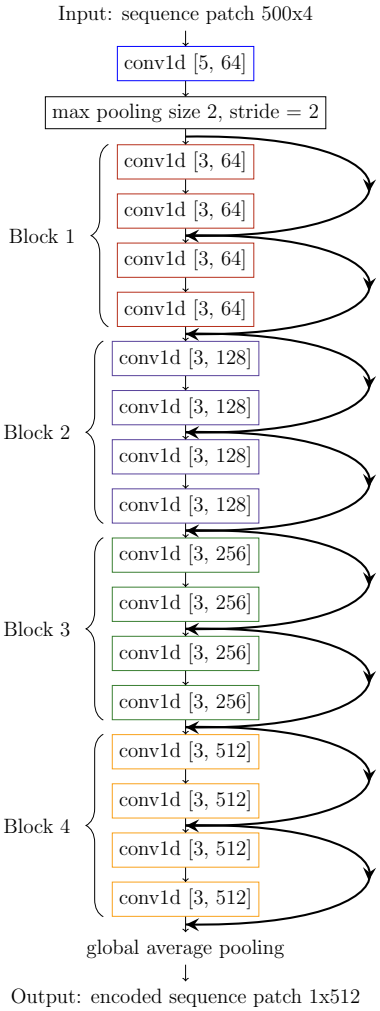

Figure A.1: Adapted one-dimensional ResNet-18 CPC encoder network architecture. The values in the brackets denote the kernel size and the channels.

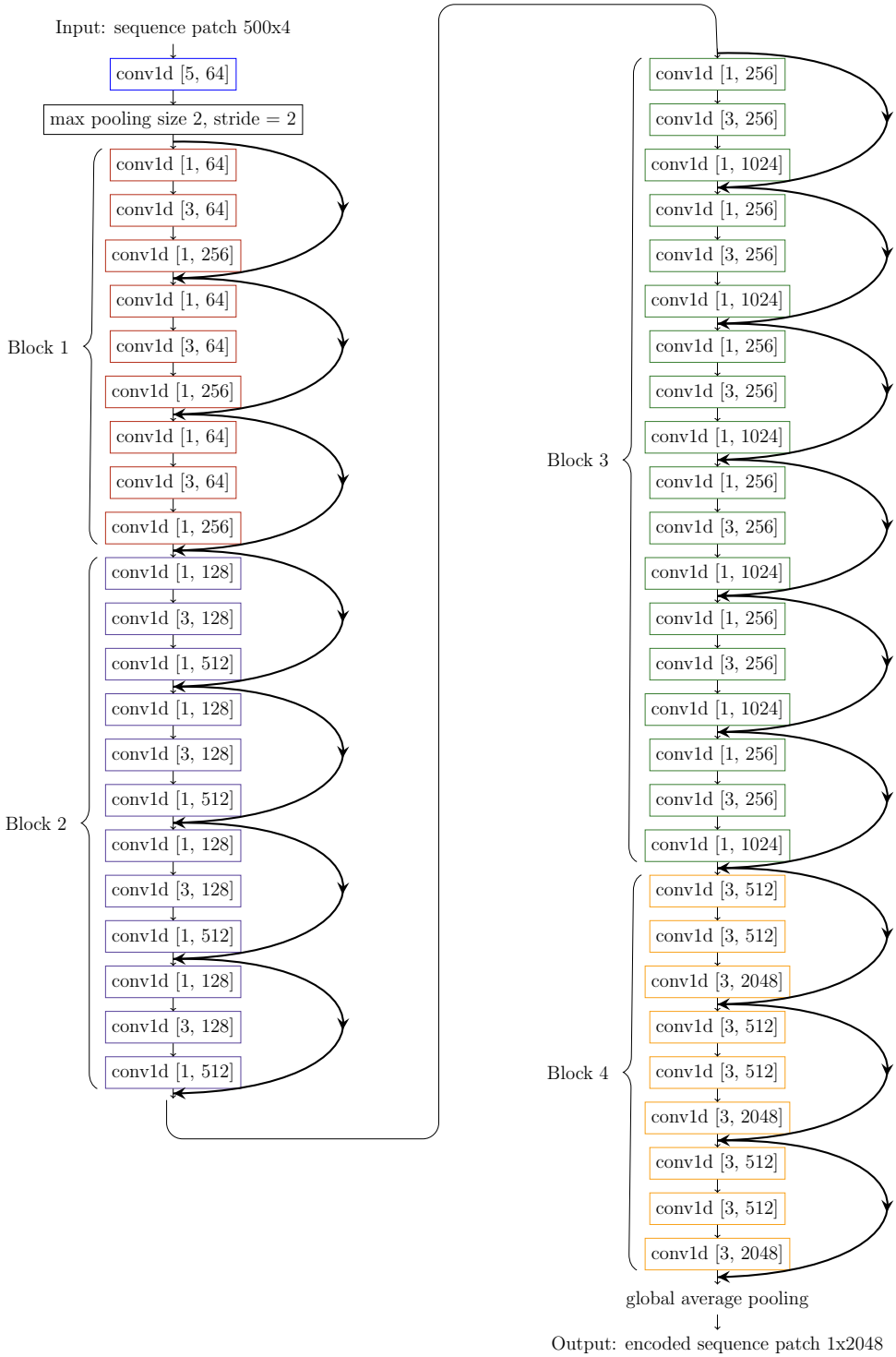

Figure A.2: Adapted one-dimensional ResNet-50 CPC encoder network architecture. The values in the brackets denote the kernel size and the channels.

Appendix B

Results

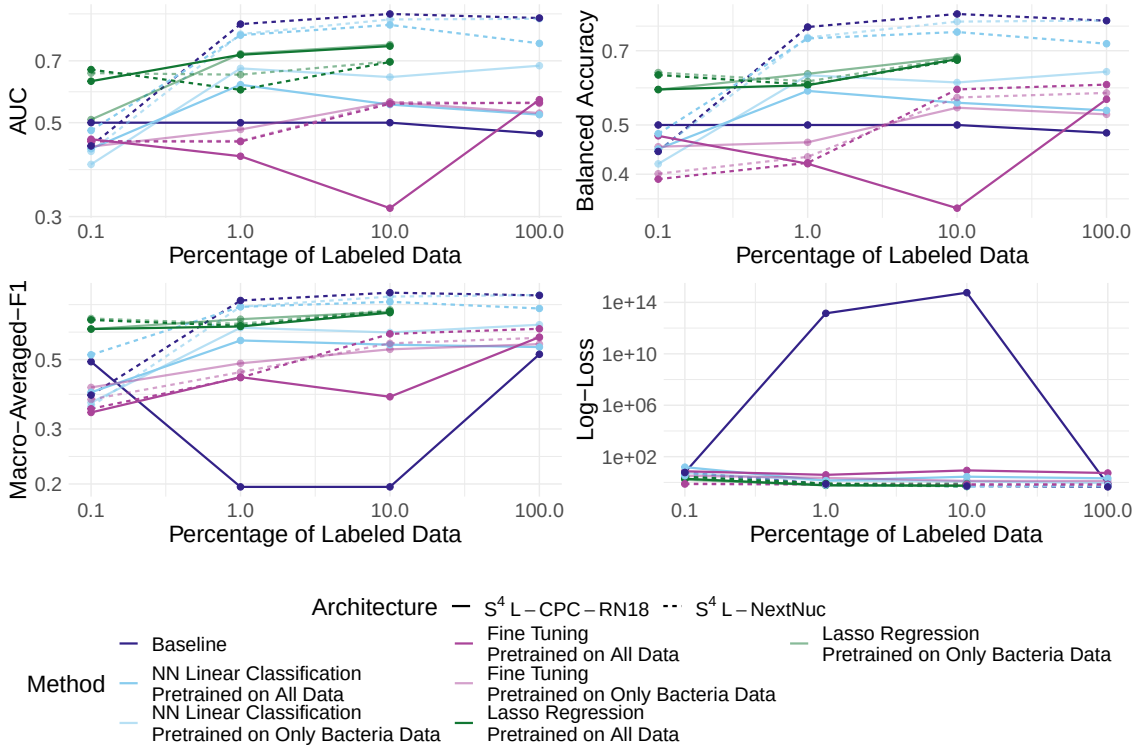

Figure B.1: Performance for Gram staining classification.

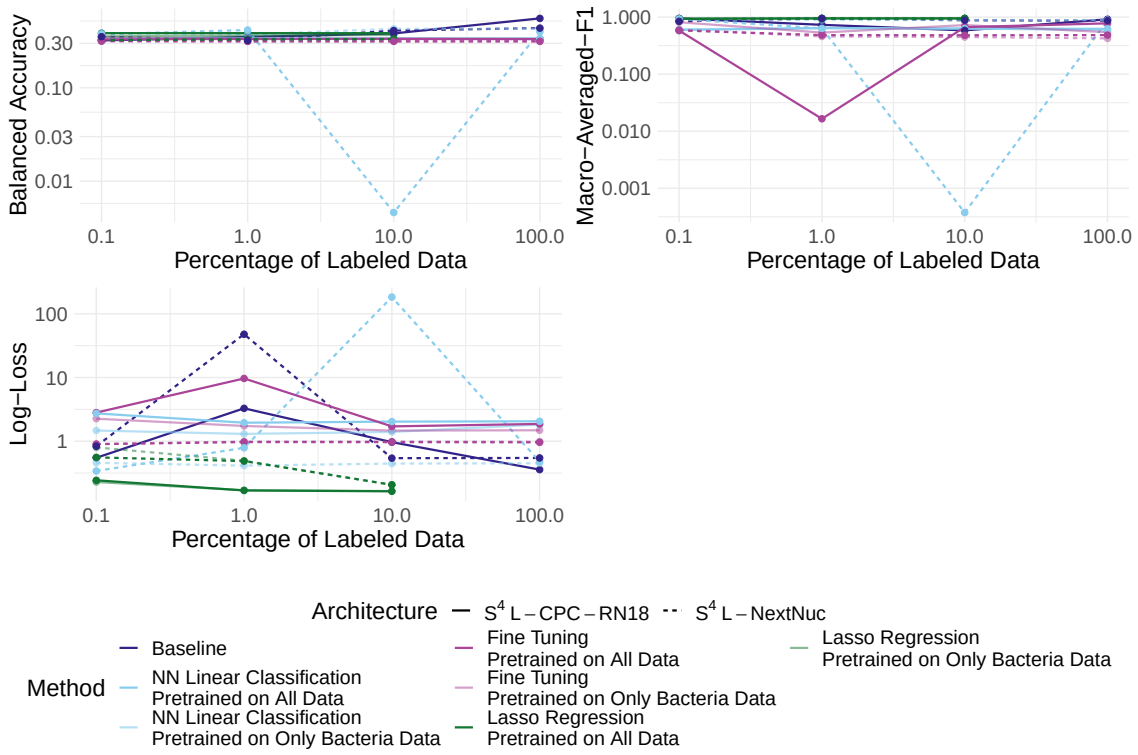

Figure B.2: Performance for Bacteria-Virus classification.

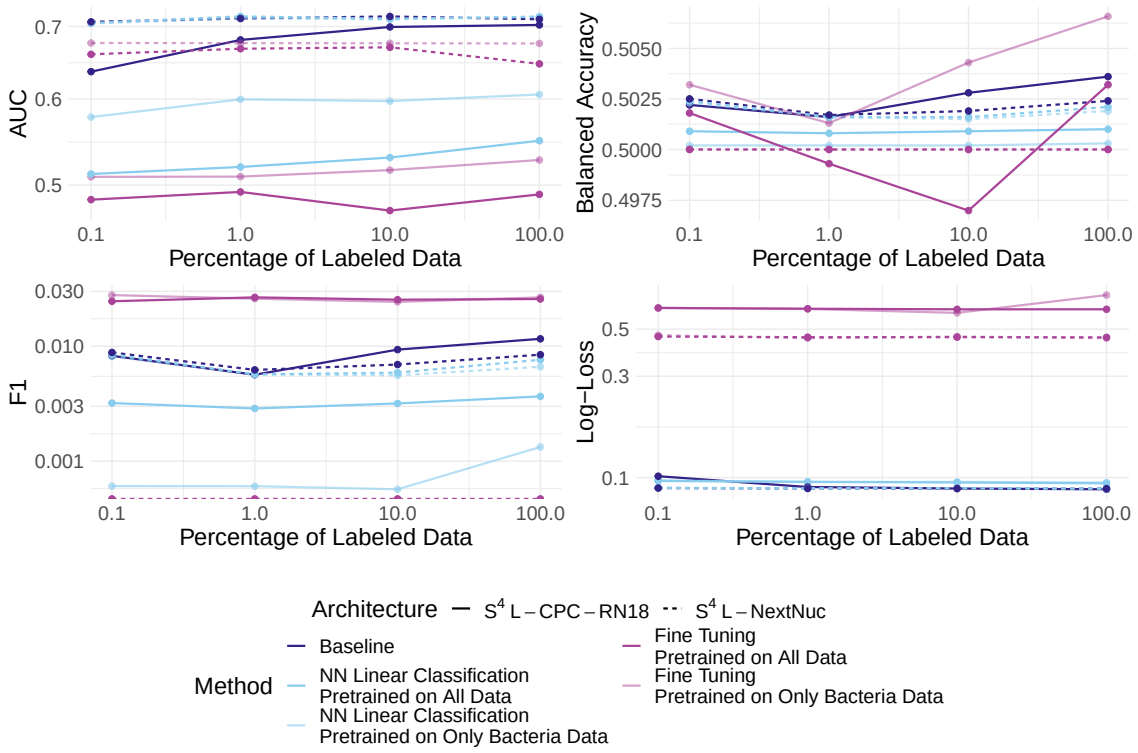

Figure B.3: Performance for chromatin features classification.

Table B.2: Detailed overview: Balanced Accuracy for Gram-Staining classification

% Data	S^4L-CPC-RN18							S^4L-NextNuc						
	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline
	L1	FT	LC	L1	FT	LC		L1	FT	LC	L1	FT	LC	
0.1	0.587	0.453	0.419	0.587	0.476	0.448	0.500	0.634	0.401	0.445	0.627	0.391	0.481	0.443
1.0	0.631	0.462	0.626	0.599	0.419	0.584	0.500	0.609	0.433	0.743	0.600	0.421	0.740	0.779
10.0	0.681	0.541	0.606	0.674	0.343	0.553	0.500	0.677	0.566	0.799	0.671	0.587	0.762	0.827
100.0		0.525	0.637		0.562	0.534	0.483		0.578	0.803		0.600	0.723	0.802

Table B.3: Detailed overview: Macro-averaged-F1 for Gram-Staining classification

% Data	S^4L-CPC-RN18							S^4L-NextNuc						
	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline
	L1	FT	LC	L1	FT	LC		L1	FT	LC	L1	FT	LC	
0.1	0.626	0.408	0.364	0.627	0.339	0.393	0.492	0.677	0.373	0.354	0.670	0.348	0.518	0.385
1.0	0.674	0.487	0.633	0.638	0.440	0.576	0.196	0.651	0.456	0.739	0.642	0.438	0.738	0.773
10.0	0.712	0.539	0.611	0.707	0.380	0.558	0.196	0.721	0.564	0.796	0.714	0.605	0.765	0.819
100.0		0.561	0.647		0.590	0.549	0.520		0.587	0.805		0.628	0.729	0.803

Table B.1: Detailed overview: Log-Loss for Gram-Staining classification

% Data	S^4L-CPC-RN18							S^4L-NextNuc						
	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline
	L1	FT	LC	L1	FT	LC		L1	FT	LC	L1	FT	LC	
0.1	1.607	3.281	5.399	1.928	7.337	15.361	5.893	1.896	0.725	5.306	2.742	0.781	3.297	5.793
1.0	0.591	2.242	1.388	0.599	3.876	1.377	10^{13}	0.807	0.697	0.699	0.814	0.701	0.625	0.739
10.0	0.551	1.307	1.187	0.557	8.582	2.787	10^{14}	0.817	0.692	0.509	0.481	0.690	0.486	0.523
100.0		1.287	0.892		5.379	2.077	0.692		0.691	0.476		0.689	0.590	0.449

Table B.4: Detailed overview: Log-Loss for Bacteria-Virus classification

% Data	S^4L-CPC-RN18							S^4L-NextNuc						
	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline
	L1	FT	LC	L1	FT	LC		L1	FT	LC	L1	FT	LC	
0.1	0.229	2.254	1.471	0.243	2.812	2.734	0.549	0.799	0.901	0.458	0.557	0.899	0.341	0.834
1.0	0.170	1.736	1.302	0.169	9.664	1.950	3.282	0.493	0.988	0.414	0.487	0.970	0.787	47.849
10.0	0.165	1.467	1.397	0.163	1.709	2.030	0.966	0.207	0.989	0.445	0.207	0.967	183.220	0.542
100.0		1.485	1.823		1.866	2.054	0.358		0.983	0.450		0.960	0.478	0.546

Table B.5: Detailed overview: Macro-averaged-F1 for Bacteria-Virus classification

% Data	S^{4}L-CPC-RN18							S^{4}L-NextNuc						
	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline	Pretrained on Bacteria Dataset			Pretrained on Combined Dataset			Baseline
	L1	FT	LC	L1	FT	LC		L1	FT	LC	L1	FT	LC	
0.1	0.942	0.813	0.610	0.939	0.581	0.595	0.949	0.920	0.602	0.906	0.924	0.584	0.928	0.844
1.0	0.948	0.536	0.640	0.950	0.017	0.643	0.733	0.940	0.458	0.890	0.942	0.484	0.635	0.949
10.0	0.949	0.726	0.628	0.950	0.667	0.630	0.586	0.944	0.448	0.890	0.944	0.481	0.000	0.881
100.0		0.545	0.593		0.778	0.623	0.916		0.424	0.887		0.484	0.877	0.866

Table B.6: Detailed overview: Log-Loss for Chromatin features classification

% Data	S^{4}L-CPC-RN18					S^{4}L-NextNuc				
	Pretrained on Bacteria Dataset		Pretrained on Combined Dataset		Baseline	Pretrained on Bacteria Dataset		Pretrained on Combined Dataset		Baseline
	FT	LC	FT	LC		FT	LC	FT	LC	
0.1	0.630	0.097	0.629	0.096	0.101	0.468	0.089	0.461	0.089	0.089
1.0	0.622	0.095	0.624	0.096	0.090	0.454	0.089	0.458	0.089	0.089
10.0	0.597	0.095	0.620	0.095	0.089	0.462	0.089	0.458	0.089	0.089
100.0	0.723	0.095	0.620	0.094	0.088	0.454	0.089	0.457	0.089	0.089

Table B.7: Detailed overview: Balanced Accuracy for Chromatin features classification

% Data	S^{4}L-CPC-RN18					S^{4}L-NextNuc				
	Pretrained on Bacteria Dataset		Pretrained on Combined Dataset		Baseline	Pretrained on Bacteria Dataset		Pretrained on Combined Dataset		Baseline
	FT	LC	FT	LC		FT	LC	FT	LC	
0.1	0.503	0.5	0.502	0.501	0.502	0.5	0.502	0.5	0.502	0.502
1.0	0.501	0.5	0.499	0.501	0.502	0.5	0.502	0.5	0.502	0.502
10.0	0.504	0.5	0.497	0.501	0.503	0.5	0.501	0.5	0.502	0.502
100.0	0.507	0.5	0.503	0.501	0.504	0.5	0.502	0.5	0.502	0.502

Table B.8: Detailed overview: F1 for Chromatin features classification

% Data	S^{4}L-CPC-RN18					S^{4}L-NextNuc				
	Pretrained on Bacteria Dataset		Pretrained on Combined Dataset		Baseline	Pretrained on Bacteria Dataset		Pretrained on Combined Dataset		Baseline
	FT	LC	FT	LC		FT	LC	FT	LC	
0.1	0.028	0.001	0.025	0.003	0.008	0	0.008	0	0.009	0.009
1.0	0.026	0.001	0.027	0.003	0.006	0	0.006	0	0.006	0.006
10.0	0.024	0.001	0.025	0.003	0.009	0	0.006	0	0.006	0.007
100.0	0.027	0.001	0.026	0.004	0.012	0	0.007	0	0.008	0.008

Table B.9: Performance results for models with replication runs.

data	Model	AUC		Log-Loss		Balanced-Accuracy		F1/macro-averaged F1	
Gram-staining	b S^4L-CPC-RN18	0.682	0.614	0.892	1.131	0.637	0.583	0.647	0.592
	a NextNuc ft 10	0.554	0.549	0.690	0.684	0.587	0.590	0.605	0.600
	a NextNuc fr 100	0.763	0.852	0.599	0.492	0.717	0.755	0.724	0.751
Bacteria Virus	a S^4L-CPC-RN18			9.664	6.789	0.335	0.324	0.017	0.401
	b NextNuc fr 1			0.414	0.607	0.373	0.414	0.890	0.822
Chromatin features	BL-NextNuc 1	0.712	0.715	0.089	0.089	0.502	0.502	0.006	0.006
	BL-CPC-RN18	0.680	0.681	0.090	0.090	0.502	0.502	0.006	0.007
	a S^4L-CPC-RN18	0.474	0.477	0.620	0.630	0.497	0.500	0.025	0.029