Modeling Interactions with Deep Learning

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy (Computer Science)

by

Jack Lanchantin

August 2021

APPROVAL SHEET

This

Dissertation

is submitted in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

Author: Jack Lanchantin

This Dissertation has been read and approved by the examing committee:

Advisor: Yanjun Qi

Advisor:

Committee Member: Vicente Ordoñez

Committee Member: Yangfeng Ji

Committee Member: Clint Miller

Committee Member: Casey Greene

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Jennifer L. West, School of Engineering and Applied Science

August 2021

Abstract

Interacting systems are highly prevalent in many real world settings, including genomics, proteomics, and images. The dynamics of complex systems are often explained as a composition of entities and their interaction graphs. In this dissertation, we design state-of-the-art deep neural networks for interaction-oriented representation learning. Learning such structural representations from data can provide highly accurate predictive models, semantic clarity, and ease of reasoning for generating new knowledge. We consider three different types of interaction graphs: 1) interactions within a particular input sample from a functional genomics task, 2) interactions between multiple input samples from a proteomics task, and 3) interactions between output labels from a computer vision task. For each type of interaction, we design novel models to tackle a real world problem and validate our results both quantitatively and visually. We show that deep learning models with relational biases can learn representations of entities and their interactions, as well as enable us to discover the governing dynamics.

Acknowledgments

First and foremost, I want to thank my advisor Yanjun Qi for her guidance and unwavering support throughout my graduate career. I am forever indebted to her for taking me on as a student and shaping me into the researcher I am today. I am very grateful for her not only mentoring me as a scientist, but also as a person. Her intelligence, curiosity, and passion for science are infectious and she has inspired me to pursue my scientific interests without boundaries. I am continually amazed by her brilliance and am incredibly thankful and proud to have worked with her for many years.

I want to extend a special thank you to Vicente Ordoñez who has been an incredible collaborator and mentor. My time working with him has been short, but I have learned an immeasurable amount. He is a fearless scientist who has had an immense impact on the way I do research and on my life in general. I want to thank Clint Miller being a great collaborator and mentor who has been a constant source of knowledge during the pandemic. I want to thank Yangfeng Ji who has showed how to frame a problem and ask the right questions. I want to thank Casey Greene who helped me start my foray into deep learning for biology and medicine and is a leader in multi-disciplinary research. I am forever grateful for all of my committee members. They are some of the smartest, humble, and courageous people I know, who have inspired me beyond words.

To my friends from home. Nick Bernardo who has been a constant source of steadiness and support. Gleason Judd who has shaped my intellectual curiosity and inspired me to pursue scientific truth. Chris Spina who inspired me to do principled work. Mike Agneta who shaped my sense of scientific exploration.

I want to give a huge thanks to my many lab-mates throughout my seven years in Virginia. My PhD work has been largely influenced by those around me, especially those in my lab. Ritambhara Singh, who has been one of my closest collaborators, but more importantly an incredible friend. She is one of the most bright, determined, and humble people I know. Weilin Xu who always kept me motivated when things weren't going well. He always has an optimistic and unique way of looking at life. Arshdeep Sekhon who has been an amazing friend, and someone I've spent many hours talking about research, philosophy, and life with. As well as Zeming Lin, Beilun Wang, Ji Gao, Zhe Wang, Dmitry Diochnos, Derrick Blakely, Jack Morris, Eli Liffland, Jake Grigsby, and Dillon Lue.

Thanks to my many amazing collaborators throughout the years including Gabe Robins, Mary-Lou Soffa, Tom Weingarten, Baoyuan Wang, Zeyu Chen, Eli Draizen, Phil Bourne, Cameron Mura, Suna Onengut-Gumuscu, and Stephen Rich. Thanks to the professors at the University of Virginia Department of Computer Science, especially Worthy Martin, Kevin Skadron, Hongning Wang, Marty Humphrey, Jason Lawrence,

Sebastian Elbaum, David Evans, Alf Weaver, Mark Floryan, Baishakhi Ray, and Malathi Veeraraghavan. Thanks also to my professors at Binghamton University, especially Mark Fowler, Scott Craver, Stephen Zahorian, Carl Betcher, Yu Chen, and Charles Westgate. I also would like to acknowledge several other people whose work was also influential in my research, some of whom I have been lucky enough to meet: Jason Weston, Rob Fergus, Arthur Szlam, Ronan Collobert, Claude Shannon, Andrej Karpathy, Anshul Kundaje, Olga Troyanskaya, Christina Leslie, William Noble, and Sepp Hochreiter. In addition, thanks to all the developers of Torch and PyTorch - this work would not have been possible without these libraries. Thanks to the department of computer science staff, as well as the Rice Hall custodial staff, especially Tony Gough and Idris Hassan.

Thanks also to all my fellow students and friends, including Garrett Bernardo, Alex Serbetzian, Aihua Chen, Sourav Maji, Avinash Kalyanaraman, Tianlu Wang, Paola Cascante-Bonilla, Elan Ashendorf, Jesse Helman, Nick Janus, Luonan Wang, Kayla Wilson, Charlotte Page, Ameer Hamdan, In Kee Kim, Qingyun Wu, Sophie Schaffeld, Kayleigh Hartigan, Noah Pannucci, Neil Aram, Dan Layman, Rob Panebianco, Paul Taukatch, Shanshan He, Fuwen Tan, Ziyan Yang, Calum d'Oelsnitz, Carl Hildebrandt, Will Leeson, Joel Mackenzie, Andrea Genovese, Rens Hoegen, Jonathan Brachthauser, the Judd family, the Bernardo family, the Agneta family, and the Spina family.

I want to thank my extended family for always being supportive and encouraging. To those who have gone, but forever are a source of my inspiration and motivation. William E. Lanchantin, who I talked to every Sunday throughout my PhD. Julia Serbalik Amodeo, who was always a light in my life. Dianne Dunn-Lanchantin, who encouraged me to pursue a PhD. As well as John Amodeo Sr., Margaret Lanchantin, Jodie Lanchantin, Wayne Morgan, and John Van Hook.

Finally, my deepest gratitude goes to my family. My Mom for showing me unconditional love, for being the most selfless and energetic person I know, for being my biggest supporter and motivator, and for always believing in me. She's my biggest source of inspiration and I could not have completed this dissertation without her. My Dad, who inspires me to seek all that's still unsung, who keeps me grounded, and who taught me that without love in the dream it will never come true. Dan for being my biggest role model, inspiring me to pursue genomics research, and reminding me to keep what's important. Alex for constantly supporting me, for inspiring me to persevere, and for encouraging me to pursue my interests.

For my parents, Kris and John.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Complex systems such as cell biology, social networks, and real world image scenes are governed by interacting entities. The intricate nature of interactive systems with extensive data like biology and images are natural targets for machine learning to not only help us accurately model them, but also understand their mechanics. Machine learning methods to simulate such systems require the correct set of modeling assumptions. In practical terms, this means designing architectures with an inductive bias that guide the model to learn the entities and interactions of the real world system. There has recently been a shift toward designing model architectures with structural biases for reasoning about data with strong interactions. Specifically, graph neural networks and self-attention based networks are designed with the assumption that interactions are important for their target tasks. Choosing and designing architectures with the right inductive biases to learn these interactions will allow for both more accurate and more interpretable predictive models. The main goal of this thesis is to study how to bridge the gap between real world interactive systems, and our ability to computationally model them and understand them. We show how using relational inductive biases within deep learning architectures can enable learning about entities, their interactions, and the rules that underlie them.

1.1 Motivating Examples

Genomics. DNA encodes the molecular language for life because it determines which set of genes are expressed in an organism. Modern sequencing methods have allowed us to read biology across the many languages of biology such as DNA, and reading this mass of data enables us to capture the complexities

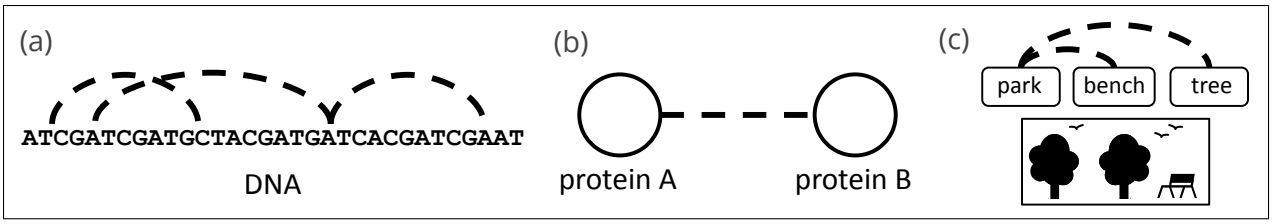

Figure 1.1: Examples of interactions. We consider three examples where interactions are crucial elements of a complex system. (a) DNA contains long range sequence interactions that determine gene the presence or absence of gene regulatory elements. (b) Proteins interact with each other in order to carry out key biological functions. (c) images contain complex interactions between objects that are present in the image.

of genomics. However, reading is not the same as understanding. This encyclopedia of information is an interconnected web of complex relations. Individual strands of human DNA contain both local and long range interactions, as illustrated in Figure 1.1 (a), which have effects on DNA properties such as gene expression. We don't yet know exactly how these sequence features and their interactions influence functional genomic signals such as transcription factor binding. Due to the vast amount of data and complex interaction structure, we argue that understanding the genome requires a computational model that can effectively learn both local sequence features as well as important long range sequence interactions.

Proteomics. While DNA is the script to carry out specific biological functions, proteins are the actors. Similar to DNA, protein sequencing methods have allowed us to read the human proteome, but this is only the first step of understanding. Protein molecules interact with each other to execute some task, forming a complex network of interactions. Yet, we don't yet understand how or why all of these interactions occur. Proteins have two "layers" of interactions: first, interactions within the amino acids, or building blocks, of the protein itself, and second, interactions between proteins, as shown in Figure 1.1 (b). Modeling this process requires a computational framework that can learn both local structural interactions within a protein as well as interactions between two proteins.

Image Understanding. Automatically understanding what is present in an image can have important outcomes across many application areas such as healthcare, retail, and security. Images in real-world applications generally portray many objects and complex situations. Predicting the set of labels corresponding to objects, attributes, or actions given an input image is a key task in visual scene recognition. The output set of labels has some structure that reflects the interaction structure of the world, as shown in Figure 1.1 (c). Effective models for image understanding require extracting good visual features that are predictive of image labels, but also exploiting the complex relations and dependencies between visual features and labels, and among labels themselves.

1.2 Thesis Overview

The dynamics of many real world complex systems can be explained by a set of entities and their interactions. Deep learning methods have proven effective at accurately modeling complex tasks. We hypothesize that deep learning models with relational biases can be used to understand and improve tasks that involve complex interactions. Allowing deep learning models to automatically identify the governing interaction patterns will help bring us from reading and processing data to understanding it. We look at prediction tasks where there are strong interactions from various viewpoints. In particular, we model interactions within DNA sequences for regulatory function classification, predict the interactions between proteins, and exploit the interactions between image labels for object classification. We introduce deep learning architectures with strong relational inductive biases that allow us to predict and manipulate interactions as well as discover interaction behaviors.

Thesis statement. *This dissertation provides novel deep learning frameworks with explicit structural assumptions to exploit, learn, and predict important interactions in complex systems. The computational predictions yielded by these frameworks suggest a number of novel hypotheses that aid in our understanding of the task dynamics.*

1.3 Contributions

Using genomics, proteomics, and image understanding as real world applications, we introduce three different frameworks for predicting and understanding the importance of interactions from various viewpoints. Namely, we consider interactions within an input, between multiple inputs, and between output labels. We show the benefits of using models that explicitly incorporate these structural interactions. The three introduced frameworks are summarized as follows.

1. **ChromeGCN.** Predictive models of DNA chromatin profile such as transcription factor binding are essential for understanding regulatory processes and developing gene therapies. It is known that long range interactions that arise from the 3D shape of DNA is highly influential in the chromatin profile. Deep neural networks have achieved strong performance on chromatin profile prediction by using short windows of DNA sequences independently. These methods, however, ignore the long-range interactions when predicting the chromatin profiles. We introduce ChromeGCN, a graph convolutional network for chromatin profile prediction by fusing both local sequence and long-range interactions. By incorporating long range interactions, we relax the i.i.d. assumption of local windows for a better representation of DNA. We show experimentally that by using ChromeGCN we get a significant improvement over

the state-of-the-art deep learning methods, particularly for DNA windows that have a high degree of interactions with other windows. Furthermore, we introduce a suite of visualization tools which allow us to identify and interpret important interactions that influence the chromatin profile, such as the relationships between DNA motifs.

2. **DeepVHPPI.** Viruses such as SARS-CoV-2 infect the human body by forming interactions between virus proteins and human proteins. However, experimental methods to find protein interactions are inadequate: large scale experiments are noisy, and small scale experiments are slow and expensive. Inspired by the recent successes of deep neural networks, we hypothesize that deep learning methods are well positioned to aid and augment biological experiments, hoping to help identify more accurate virus-host protein interaction maps. Moreover, computational methods can quickly adapt to predict how virus mutations change protein interactions with the host proteins. Previous sequence-based computational methods do not incorporate proper structural information into their framework, resulting in models that may not generalize well. We propose DeepVHPPI, a novel deep learning framework combining a self-attention-based transformer architecture and a transfer learning training strategy to predict interactions between human proteins and virus proteins that have novel sequence patterns. We show that our approach outperforms the state-of-the-art methods significantly in predicting Virus–Human protein interactions for SARS-CoV-2, H1N1, and Ebola. In addition, we demonstrate how our framework can be used to predict and interpret the interactions of mutated SARS-CoV-2 Spike protein sequences

3. **C-Tran.** Multi-label image classification is the task of predicting a set of labels corresponding objects or entities present in an image. Correctly modeling the interactions between objects is a key component in visual scene understanding. We propose the Classification Transformer (C-Tran), a general framework for multi-label image classification that leverages Transformers to exploit the complex interactions among visual features and labels. Our approach consists of a Transformer encoder trained to predict a set of target labels given an input set of masked labels, and visual features from a convolutional neural network. A key ingredient of our method is a label mask training objective that uses a ternary encoding scheme to represent the state of the labels as positive, negative, or unknown during training. Our model shows state-of-the-art performance on challenging datasets such as COCO and Visual Genome. Moreover, because our model explicitly represents the uncertainty of labels during training, it is more general by allowing us to produce improved results for images with partial or extra label annotations during inference. We demonstrate this additional capability in the COCO, Visual Genome, News500, and CUB image datasets.

Chapter 2

Background

2.1 Deep Learning Models and Relational Inductive Biases

Not only are many real world environments rich in interactive structure, but also the way we think about them: humans reason and communicate complex systems in terms of entities and their interactions [8]. When learning, humans either fit new knowledge into our existing interaction models, or adjust the model itself to better satisfy previous knowledge [1, 9, 10, 11]. A key principle of human intelligence is combinatorial generalization, or constructing new predictions from known elements that can interact to compose limitless systems [1, 12, 13]. Human combinatorial generalization is based on our abilities to represent structure and reasoning about interactions. We can take a set of words and understand how they create meaning through their interactions. We can observe objects and understand visual scenes as a composition of their interactions. A key component of artificial intelligence moving forward involves teaching computational models to perform combinatorial generalization.

Recently, model architectures with structural biases have shown convincing empirical results for reasoning about data with strong interactive properties [14, 15, 16, 17, 18, 1, 19]. These methods perform computation over entities and their relationships, where the structure of the entities and the importance of interactions between them are learned rather than specified. Relational inductive biases, in the form of specific architectural assumptions, guide these approaches towards learning about entities and relations [1].

The inductive bias of a machine learning model is the set of assumptions that the model uses to predict outputs of given inputs that it has not encountered [20]. When considering a system that contains complex interactions, the inductive bias of a model is an important property in order to properly learn the interactions.

Component	Entities	Relations	Relational inductive bias	Invariance
Fully Connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph Network	Nodes	Edges	Arbitrary	Node, edge permutations
Transformer	Tokens	All-to-all	Arbitrary	Token permutations

Table 2.1: Extended from Battaglia et al. [1], we summarize the five main deep learning components including what types of entities and relations they model, as well as the inductive biases and invariances they encode. We note that Transformers (self-attention) can incorporate specific relations between tokens, but in their general form they learn the important relations during training.

In this section, we give a tour of commonly used deep learning model components and the inductive biases each takes on. We summarize the five main components in Table 2.1. Importantly, the convergence of graph neural networks and Transformers, which are generalizations of previous deep learning architectures, facilitate the learning of key interactions from large datasets.

2.1.1 Fully Connected Networks

Fully connected neural networks [21] are an affine transformation on input $\mathbf{x}$, followed by an added bias term, and finally a non-linearity:

$$\mathbf{z} = \sigma(\mathbf{B}_i + \mathbf{W}\mathbf{x}), \tag{2.1}$$

where $\mathbf{W}$ and $\mathbf{B}$ are the trainable parameters of the network, and σ is an element-wise non-linearity, such as the rectified linear unit (ReLU): $\text{ReLU}(x) = \max(0, x)$. The interactions are thus modeled as all-to-all (all units in the output are connected to all units in the input), and the rules are specified by the weights and biases. All input entities interact with each other to determine the output, and therefore there is a weak inductive bias.

2.1.2 Convolutional Neural Networks

Convolutional neural networks (CNNs), first introduced in the computer vision setting, were designed to model local, translation invariant features [22]. These work on input domains where there is a local spatial structure to the domain. For example, natural images often contain objects that have local spatial structures which determine that particular object. A 1D temporal convolution with filter (or kernel) size k takes an input data matrix $\mathbf{X}$ of size $T \times n_{in}$, with length T and input layer size n_{in}, and outputs a matrix $\mathbf{X}$ of size

$T \times n_{out}$, where n_{out} is the output layer size:

$$\mathbf{z}_{t,i} = \sigma(\mathbf{B}_i + \sum_{j=1}^{n_{in}} \sum_{z=1}^{k} \mathbf{W}_{i,j,z} \mathbf{x}_{t+z-1,j}), \tag{2.2}$$

where $\mathbf{W}$ and $\mathbf{B}$ are the trainable parameters of the convolution filter, and σ is a function enforcing element-wise nonlinearity. Multiple convolutional layers are typically used in sequence to learn different feature representations. The introduced 1D temporal convolution can be easily extended to 2D spatial convolutions.

In CNNs, model parameters are shared across input subregions, providing the model with an inductive bias to learn features that are translation invariant. Therefore, they learn local features that can appear anywhere in an input.

2.1.3 Recurrent Neural Networks

While convolutional networks are designed to model spatial input features, in many settings the inputs can also be sequential (e.g. video, text or speech). Designed to handle sequential data, Recurrent neural networks (RNNs) have become the commonly used model for tasks such as natural language understanding [23, 24]. The key advantage of RNNs over CNNs is that they are able to find long range patterns in the data which are highly dependent on the ordering of the sequence for the prediction task.

Given an input matrix $\mathbf{z}$ of size $T \times n_{in}$, an RNN produces matrix $\mathbf{Z}$ of size $T \times d$, where d is the RNN embedding size. At each timestep t, an RNN takes an input column vector $\mathbf{x_t} \in \mathbb{R}^{n_{in}}$ and the previous hidden state vector $\mathbf{z_{t-1}} \in \mathbb{R}^d$ and produces the next hidden state $\mathbf{z_t}$ by applying the following recursive operation:

$$\mathbf{z}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{z}_{t-1} + \mathbf{b}), \tag{2.3}$$

where $\mathbf{W}, \mathbf{U}, \mathbf{b}$ are the trainable parameters of the model, and σ is an element-wise nonlinearity.

Due to their recursive nature, RNNs can model the full conditional distribution of any sequential data and find dependencies over time, where each position in a sequence is a timestep on an imaginary time coordinate running in a certain direction. Weight sharing is done over time steps, providing the model with a sequential inductive bias, allowing them to generalize on sequential data. RNNs can therefore learn sequential interactions between input tokens, and also encode a locality bias in the sequence through their Markovian structure [1].

2.1.4 Graph Neural Networks

Graph neural networks (GNNs) [25, 16, 17] model elements as nodes on a graph G. Here $G = (V, E)$, where V describes the set of nodes (variables) and E denotes the set of edges (about how variables interact with other variables). GNNs are a generalization of CNNs where there is no explicit spatial structure known, but a graph structure is known. In GNNs, each node $\mathbf{z}_i$ is updated using its neighboring nodes $\mathbf{z}_j, j \in \mathcal{N}(i)$, where $\mathcal{N}(i)$ denotes the neighbors of node i obtained from $\mathbf{A}$. The GNN works by revising a window's representation $\mathbf{z}_i^t$. We denote t to represent the G layer index. In the most basic graph neural network, each $\mathbf{z}_i^t$ is revised using a parameterized summation of neighbors, $\mathbf{z}_j^t, j \in \mathcal{N}(i)$:

$$\mathbf{z}_i^{t+1} = \sigma\left(\frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{z}_j^t \mathbf{W}^t\right), \tag{2.4}$$

where $\sigma(\cdot)$ is a non-linear activation function such as tanh or ReLU and $\mathbf{W}^t \in \mathbb{R}^{d \times d}$ is a linear feature transformation matrix for the GNN layer t. Notably, using the summation in Eq. 2.4, the representation of each variable $\mathbf{z}_i$ is updated based on the representation of its neighbors. More complex node update functions are commonly used, such as attention-based [26]. After T rounds of iterative updates to spread information to distant nodes, a readout function R is used on the updated node embeddings to make predictions node classification, edge (link) classification, or graph classification.

Graph neural networks work on non euclidean data, and thus are equipped with an arbitrary inductive bias. The structure (i.e. edges) of the graph largely determines the types of interactions the model can learn. Therefore, a well defined graph can lead the model to learn the important interactions between entities.

2.1.5 Transformers

A Transformer [19] $f_\theta : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$ is "transforms" a collection of n objects in $\mathbb{R}^d$ to another collection of n objects in $\mathbb{R}^d$ [27]. Transformers were initially applied to sequential data in the context of NLP, but because they encode relational structure as data they admit straightforward application to data with non-linear relational structure. A transformer block is a parameterized function mapping $f_\theta : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$. If $\mathbf{x} \in \mathbb{R}^{n \times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$, where

$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \quad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d \times k} \tag{2.5}$$

$$\alpha_{i,j}^{(h)} = \text{softmax}_j \left(\frac{\left\langle Q^{(h)}\left(\mathbf{x}_i\right), K^{(h)}\left(\mathbf{x}_j\right) \right\rangle}{\sqrt{k}} \right) \tag{2.6}$$

$$\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j} V^{(h)}\left(\mathbf{x}_j\right), \quad W_{c,h} \in \mathbb{R}^{k \times d} \tag{2.7}$$

$$\mathbf{u}_i = \text{LayerNorm}\left(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1\right), \quad \gamma_1, \beta_1 \in \mathbb{R} \tag{2.8}$$

$$\mathbf{z}_i' = W_2^T \sigma\left(W_1^T \mathbf{u}_i\right), W_1 \in \mathbb{R}^{d \times m}, W_2 \in \mathbb{R}^{m \times d} \tag{2.9}$$

$$\mathbf{z}_i = \text{LayerNorm}\left(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2\right), \quad \gamma_2, \beta_2 \in \mathbb{R} \tag{2.10}$$

where σ is a nonlinearity such as ReLU, LayerNorm is a normalization technique [28] with parameters γ, β, and the W matrices are the learned weights of the network. d, k, m, H, and L are hyperparameters of the model. The interactions or relationships between input tokens are learned via neural attention (α). Neural attention was introduced to allow deep learning models to learn the dependencies between tokens or features [29]. Tokens communicate with each other and update their representations by their learned attention weights. Importantly, attention allows us to learns interaction importance when it is not known or predefined

Transformers are a generalization of GNNs where the graph is typically fully connected and the interaction importance is learned via attention. While Transformers are similar to fully connected networks in that they allow all input entities interact with each other, they learn the relational structure between any amount of entities via attention, and thus can encode arbitrary relationships. Transformers are the most general model out of the five listed because they enforce the least inductive bias.

2.2 Genomics and Proteomics

Biology is extensively governed by sequence information. Billions of nucleotide characters encode the instructions of the human genome, and millions of amino acid characters encode the instructions of the human proteome. These strings of characters, similar to natural language, denote the code of life. While modern sequencing methods make this information increasingly available, we cannot currently understand exactly what these strings of characters mean, and how they interact with each other to regulate biological processes. Many processes related to biological sequences contain very long range interactions within a molecule, and interactions between molecules. Below we describe the different types of biological sequence data in detail.

2.2.1 Genomics

DNA Sequences. Deoxyribonucleic Acid (DNA) is the building block of organisms. It consists of the information required by a cell to function properly. The double-helix structure of DNA, consisting of two strands, is made of four nucleotide bases: Adenine (A), Guanine (G), Thymine (T) and Cytosine(C). 'DNA sequencing' is the process of determining the precise arrangement of these bases using machines. This sequence of characters contains the information [30] needed to describe the genome. Inside each cell, a DNA molecule (3 billion letters bp long) is broken into 23 smaller sections called chromosomes which contain sub-sections called 'genes' that store the regulation information. When a gene is expressed, a protein is created, and when a gene is repressed, a protein is not created. The 23 pairs of chromosomes consist of about 70,000 genes, and every gene has its function. An important aspect about DNA is that its tightly compact nature leads to interactions that are very long in the 1-dimensional sequential space, but very close in the 3-dimensional compact space. These interactions are crucial for the regulation of certain genes.

Chromatin Profile Signals. While the DNA provides the code for gene expression regulation, there are many signals that determine regulation. For example, Transcription Factors (TFs) are proteins that bind to sequence-specific locations on the DNA near the gene and initiate the process of DNA code conversion to proteins. Histone modifications are chemical changes to the proteins that DNA wraps around, leading to regulation changes. Both of these events are captured digitally by using Chromatin immunoprecipitation sequencing, or ChIP-seq. Chromatin immunoprecipitation allows us to separate DNA segments that are involved in the protein activities. We then perform DNA sequencing to get the sequences of these DNA segments. Accessible DNA is open regions (i.e. where there are not proteins that the DNA is wrapped around), that allow a gene to be expressed. This is captured via ATAC-seq. We call the regulatory mechanisms that help determine gene expression the 'chromatin profile'.

2.2.2 Proteomics

Protein Sequences. When a gene is expressed, the sequence information of the genes gets converted into proteins via transcription and translation. Proteins are essential biomolecules that are involved in almost every process inside a cell ranging from oxygen transport, cell signaling, and immunity. Proteins are a chain of molecules called 'amino acids' that are linked by covalent bonds. There are 20 different amino acids, allowing us to represent proteins as a string with an alphabet of 20 characters. Each protein can be viewed as a string

that's typically around 500 characters long. Protein sequencing is the practical process of determining the precise order of amino acids in a protein using a machine.

Protein Structure and Interactions. A key aspect about proteins is that the string of amino acids folds first into local secondary structures and then into its full 3D tertiary structure. And this structure is what determines the function, or role of a protein. But proteins rarely act alone. Instead they interact with other proteins in order to carry out specific functions. So in order to determine what a protein really does, we must first know who it interacts with [31]. This is critically important especially when studying viruses because viruses infect humans through protein-protein interactions.

2.3 Multi-label Image Classification

The objective of computational image recognition is mimic the recognition capabilities of human vision using computational methods [32]. Understanding the content of an image depends on a rich understanding of objects and their interactions. Several works have used relational deep learning models to learn the interactions between objects in images on simulated datasets [33, 34]. Multi-label image classification is the task of predicting all objects present in a real world image. This requires modeling the strong relationship structure of the objects, or labels. Multi-label classification has a rich history in text [35, 36], images [37, 38], bioinformatics [37, 38], and many other domains.

Formally, let $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$ be the set of data samples with inputs $\boldsymbol{x} \in X$ and outputs $\boldsymbol{y} \in Y$. Inputs $\boldsymbol{x}$ are a (possibly ordered) set of S components $\{x_1, x_2, ..., x_S\}$, and outputs $\boldsymbol{y}$ are a set of L labels $\{y_1, y_2, ..., y_L\}$. Multi-label classification involves predicting the set of binary labels $\{y_1, y_2, ..., y_L\}, y_i \in \{0, 1\}$ given input $\boldsymbol{x}$. In images, there are strong interaction relationships between the output labels $\{y_1, y_2, ..., y_L\}$. Modeling these relationships is a key component of an intelligent visual recognition model. Most methods rely on learning these dependencies through feature representation learning [39, 40, 41, 42]. These methods do not allow for an intuitive way to understand the relationships that the model has learned, as well as a way to interact with, and test relationships.

The naïve approach to multi-label prediction is to predict all labels independently of one another, assuming no dependencies among labels. Binary relevance methods predict each label separately as a logistic regression classifier for each label [41, 42]. That is, they use the following conditional probability parameterized by learned weights W:

$$P(Y|X; W) = \prod_{i=1}^{L} p(Y_i|X_{1:S}; W) \tag{2.11}$$

However, incorporating the interactions, or joint dependencies, between labels and input features are crucial. We therefore want a model that can represent or approximate the following joint distribution:

$$P(Y|X;W) = \prod_{i=1}^{L} p(Y_i|Y \backslash Y_i, X_{1:S}; W) \tag{2.12}$$

Exploiting these interaction dependencies can lead to both more accurate models and interpretable results that can help us learn the relationships between features and labels.

Chapter 3

Modeling Genomic Sequence Interactions with ChromeGCN

Predictive models of DNA chromatin state such as transcription factor binding are important for modeling how small sequence mutations can effect the state, as well as attribute predictions to specific elements and interactions. It is well known that the arrangement of nucleotides, or language, of DNA is responsible for the chromatin state. However, the interactions between subsequences of DNA are vital to the specific instructions being executed. Correctly modeling these interactions is therefore a key component of an accurate and interpretable chromatin state prediction model.

Research Question 1: *Can we model both local sequence features and long range interactions for chromatin profile prediction?*

In this chapter, we introduce ChromeGCN, a graph convolutional network for chromatin state prediction by combining both local sequence and long-range interaction information. The inductive bias of ChromeGCN allows for more accurate predictions and easy interpretation of important genomic interactions. Furthermore, we introduce the Deep Motif Dashboard, a framework to visualize local DNA sequence features and interactions.

3.1 Introduction

The human genome includes over 3 billion base pairs (bp), each being described as A,C,G, or T. Chromatin (DNA and its organizing proteins) is responsible for many regulatory processes such as controlling the expression of a certain gene. Active chromatin elements such as transcription factors proteins binding at

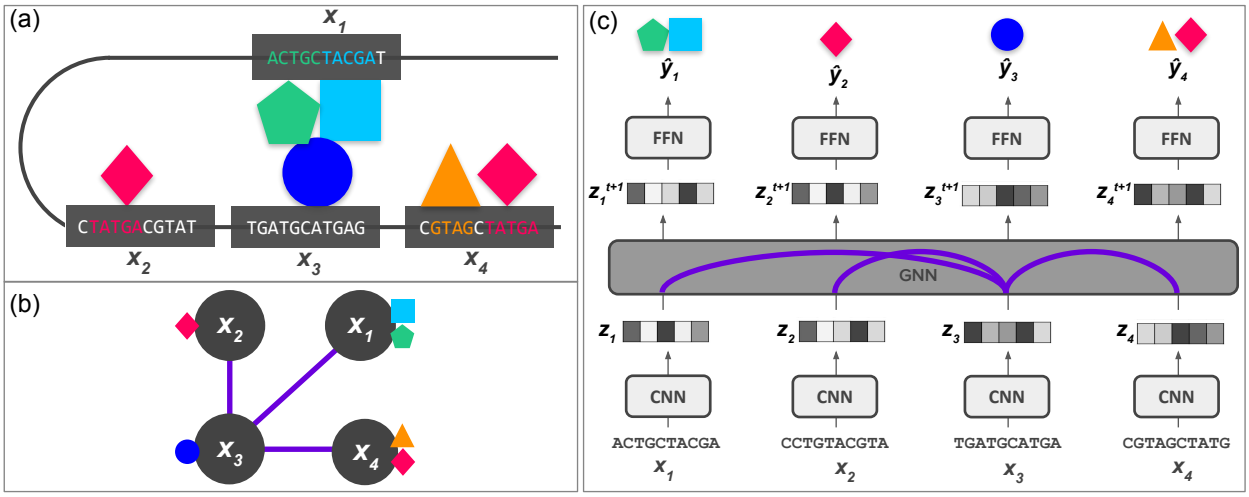

Figure 3.1: **(a) 3D Genome.** The 3D shape of chromatin can lead DNA "windows" (shown in grey boxes) far apart in the 1D genome space to be spatially close. These spatial interactions can influence chromatin profiles, such as TFs binding (as shown by the colored shapes). In most cases, the DNA sequence determines the chromatin profile. However, it can also be influenced by interactions, such as the formation of TF complexes shown in the middle. **(b) Graph Representation of DNA.** Using Hi-C data, we can represent subfigure (a) using a graph, where the lines between windows are the edges indicated by Hi-C data. **(c) ChromeGCN.** By using a graph convolutional network on top of convolutional outputs the model considers the known dependencies between long-range DNA windows. The lines between windows correspond to edges in Hi-C data.

particular location in DNA or histone modifications are what constitute that location's "chromatin profile". Understanding the chromatin profile of a local region of DNA is a step toward understanding how that region influences relevant gene regulation since chromatin profile is a direct factor in regulating expression. Since biological experiments are time-consuming and expensive, computational methods that can accurately simulate and predict the chromatin profile are crucial. Modeling the chromatin profile of each base pair has been a long standing challenge due to the sheer length and complexity of genome DNA. Deep neural networks have shown state-of-the-art performance in extracting useful features from segments of DNA to predict the chromatin profile (e.g., if a transcription factor protein binds to that location or not) [43, 44]. However, these methods heuristically divide DNA into local "windows" (e.g., about 1000bp long) and predict the states of each window independently, disregarding the effects of distant windows. Due to the spatial 3D organization of chromatin in a cell, distal DNA elements (potentially over 1 million bp away) have shown to have effects on chromatin profiles [6, 45, 46].

Fig. 3.1(a) illustrates the importance of using both sequence and 3D genome data. This figure shows long-range dependencies between chromatin windows, where the colored shapes represent multiple transcription factors (TF) proteins. TFs typically bind to specific sequence patterns in DNA known as motifs [47]. However,

a TF may also bind to a DNA window due to the presence of other TFs nearby in the 3D space because they form a protein complex [48, 46]. Such a case will result in motifs far away in the 1D genome coordinate space, but nearby in the 3D space. The corresponding dependencies between the chromatin windows are illustrated by the triangle, square, and circle TFs. The two segments interacting in the middle of the diagram are very far in the 1D sequence representation (represented by the grey line), but very close in the 3D representation. Similarly, a TF may not bind to a segment with its motif present due to another interfering TF nearby in the 3D space. These types of interactions are lost in data-driven prediction models that only consider local DNA segments independently.

However, modeling these known long-range interactions between windows is difficult. Local sequence window-based prediction methods assume data samples are independent according to the commonly used independent and identically distributed (IID) assumption. Yet, the long-range dependencies existing in DNA make windows not IID.

Modeling long-range, or non-local interactions, has had a long history in many areas such as natural language processing, where the label of one particular segment depends on the label of a segment far away. Recurrent neural networks such as LSTMs [49] have been used to model non-local dependencies where the model relies on the hidden state to remember the state of a token (e.g., a word) very far away. However, LSTMs are known to only remember a small number of tokens back, leading to rather "local" relationship learning [49, 19]. This drawback has lead to an increasing interest in the explicit modeling of non-local dependencies via pairwise interaction models such as transformers [19, 50, 51].

In a related line of work, graph convolutional networks (GCNs) have been proposed to model the pairwise dependencies of nodes in graph or 3D structured data such as citation networks and point clouds [14, 16, 52, 53]. This direct modeling of edges allows the network to learn non-local relationships. While typically viewed in its 1D sequential form, DNA can be represented as 3D genome structured data via Hi-C maps, as shown in Fig. 3.1(b). Hi-C maps are matrices that give the number of contacts between two segments of DNA, and normalized Hi-C maps tell us the likelihood of two locations interacting [54]. Using Hi-C data, segments of DNA can be represented as nodes on a graph, and edges are interactions between segments. Such interactions can be crucial in regulatory processes such as gene transcription [6]. That is, Hi-C contacts are a direct reflection of how distant chromatin elements interact. We hypothesize that accounting for such interactions will lead to improved chromatin profile prediction accuracy.

In this work, we propose ChromeGCN, a novel method that uses a fusion of both sequence and 3D genome data (in the form of Hi-C maps) to predict the chromatin profile of DNA segments. To the best

of our knowledge, ChromeGCN is the first deep learning framework that successfully combines sequence and 3D genome data to model both local sequence features and long-range dependencies for chromatin profile prediction. ChromeGCN works by first representing DNA windows as a d-dimensional vector with a convolutional neural network on the local window sequence. We then revise the window vector using a graph convolutional network on all window relationships from Hi-C 3d genome data. We test ChromeGCN on datasets from two cell lines where we compare against the previous state of the art chromatin profile prediction methods. We demonstrate that ChromeGCN outperforms previous methods, especially for labels that are highly correlated with long-range chromatin interactions.

An important aspect of ChromeGCN is that it allows us to better understand the chromatin profile in terms of genomic features and interactions. Using ChromeGCN, we propose Hi-C saliency maps to understand which Hi-C contacts are most important for chromatin profile labeling. Since ChromeGCN uses explicit long-range relationships from Hi-C data (as opposed to implicit long-range relationships using a recurrent neural network), we can easily understand the important relationships for greater interpretability. Furthermore, we introduce the Deep Motif Dashboard to investigate important local sequence features within a particular genomic region that influence the chromatin state.

The main contributions of this chapter are:

1. We propose ChromeGCN, a novel framework that incorporates both local sequence and long-range 3D genome data for chromatin profile prediction.

2. We experimentally validate the importance of ChromeGCN on two cell lines from ENCODE, showing that modeling long range genome dependencies is critically important.

3. We introduce Hi-C saliency maps, a method to identify the important long range interactions for chromatin profile prediction from Hi-C data, and the Deep Motif Dashboard, a framework to visualize local sequence features and interactions.

3.2 Background and Related Work

3.2.1 Predicting Chromatin Profile Using Machine Learning

Computational models for accurately predicting chromatin profile labels from DNA sequence have gained popularity in recent years due to the urgency of the task for many applications. For instance, predicting how

chromatin effects vary when variants in DNA occur. The importance of computational modeling arises from the low cost and high speed in comparison to biological lab experiments.

One class of methods for state prediction used generative models in the form of position weight matrices (PWM) [47]. These methods construct motifs, or short contiguous sequences (often 8-20bp in length), which are representative of a particular chromatin profile label such as a transcription factor binding. A new sequence can then be classified according to how well it matches the motif. A significant drawback of using predefined motif features is that it is difficult to find the correct motifs for predicting unseen sequences [44]. Another class of methods use string kernels (SK) [55, 56], where some kernel function is built to capture the similarity between DNA segments according to substring patterns. However, these methods suffer from the issue of a predefined feature engineering. Moreover, these methods do not scale to a large number of sequences [43].

To overcome the issues of PWM and SK methods, researchers turned to automatic feature extraction using deep neural networks which have outperformed both generative PWM and SK methods [44, 43]. Convolutional neural networks (CNNs) were the first deep learning method to outperform previous methods. CNNs have been used extensively to learn features of DNA for sequence-based prediction [43, 44, 57, 58, 7]. The benefit of convolutional models is that they have an inductive bias for modeling translation invariant features in DNA sequences. This allows CNNs to effectively learn the correct "motifs" or kernels for chromatin labeling. There has since been several revisions to the original CNN models for marginally better feature extraction, such as adding a recurrent network on top of the CNN motif features [59, 60].

However, current state-of-the-art models only learn the features from the sequences of individual local windows and not between windows (i.e., longer-range interactions). Since DNA interacts with itself in the form of long-range 3D contacts, labeling the chromatin profiles of a window can be affected by another distant window. [61] use longer range dependencies (32kbp), but the dependencies are modeled implicitly using dilated convolution across 128bp windows. Accordingly, methods that account for explicit long range 3D chromatin contacts are needed to model the true interactions in DNA.

3.2.2 DNA Interactions from Hi-C Maps

Hi-C experiments, and 3C experiments in general, are biological methods used to analyze the spatial organization of chromatin in a cell. These methods quantify the number of interactions between genomic loci. Two loci that are close in 3D space due to chromatin folding may be separated by up to millions of

nucleotides in the sequential genome. Such interactions may result from biological functions, such as protein interactions [62].

Since the first Hi-C maps were generated, many works have been introduced to analyze the maps. [46] investigated the spatial relationships of co-localized TF binding sites within the 3D genome. They show that for certain TFs, there is a positive correlation of occupied binding sites with their spatial proximity in the 3D space. This is especially apparent for weak TF binding sites and at enhancer regions.

[63] identified that the ZNF143 TF motif in the promoter regions provides sequence specificity for long range promoter-enhancer interactions. [64] identified coupling DNA motif pairs on long-range chromatin interactions. [65] use convolutional neural networks to predict Hi-C interactions from sequence inputs. None of the previous methods, however, use known Hi-C data to learn better feature representations of genomic sequences for chromatin profile prediction.

3.2.3 Graph Convolutional Networks

Graph convolutional networks (GCNs) were recently introduced to model non-local or non-smooth data [14, 66, 16, 67, 17, 26]. For the task of node classification, GCNs can learn useful node representations which encode both node-level features and relationships between connected nodes. Essentially, GCNs learn node representations by encoding local graph structures and node attributes, and the whole framework can be trained in an end-to-end fashion. Because of their effectiveness in learning graph representations, they achieve state-of-the-art results in node classification. The main assumption is that the input samples (in our case, individual DNA windows) are not independent. By modeling the graph dependency between samples, we can obtain a better representation of each of the samples. Non-local neural networks [52] are an instantiation of graph convolution, which was designed to model the long-range interactions in video frames.

3.3 Problem Formulation and Data Processing

The objective of chromatin profile prediction (i.e., chromatin effect prediction) is to tag segments of DNA with the probability of how likely a certain chromatin effect (aka chromatin profile label) is present. In our formulation, we define chromatin profile labels to include transcription factor (TF) binding, histone modifications (HM), and accessibility (DNase I). This is known as a multi-label classification task, where multiple labels can be positive at once (different from multi-class tasks where only one label can be positive).

Formally, given an input DNA window $\mathbf{x}_i$ (a segment of length T), we want to predict $y_i^l \in \{0, 1\}$ for a label l, where l ranges from 1 to L.

Cell Line	Train Windows	Valid Windows	Test Windows	TFs	HMs	DNase I
GM12878	368,082	89,911	79,731	90	11	2
K562	457,609	106,777	117,815	150	12	2

Table 3.1: **Datasets Summary.** GM12878 contains 103 total chromatin profile labels, and K562 contains 164 total labels. We use the same chromosomes for training, validation, and testing for both datasets.

Sequence Data. We derive chromatin labels using ChIP-seq data from ENCODE [68]. We use the cell lines GM12878 and K562, two of the most widely used from ENCODE and Roadmap [68, 69]. For each cell line, we use all windows which have at least one positive chromatin ChIP-seq peak. We consider any peak from ENCODE to be a positive peak. We follow a similar setup as in [43] where we bin the DNA into 1000bp windows. If any ChIP-seq peak overlaps with at least 100bp of a particular window, we consider that a positive window for that chromatin label. We then extract the 2000bp sequence surrounding the center of each window as the input features, as done in [7], since the motif for a particular signal may not be contained fully in the 1000bp length window. Although we use the 2000bp sequence, we consider each window to be the original non-overlapping 1000bp for notation purposes. An illustration of how sequences are extracted is shown in Fig. 3.2 (a). Following [43], we use chromosome 8 for testing and also add chromosomes 1 and 21. Chromosomes 3, 12, and 17 are used for validation, and all other chromosomes (excluding X and Y) are used for training. The datasets are summarized in Table 3.1.

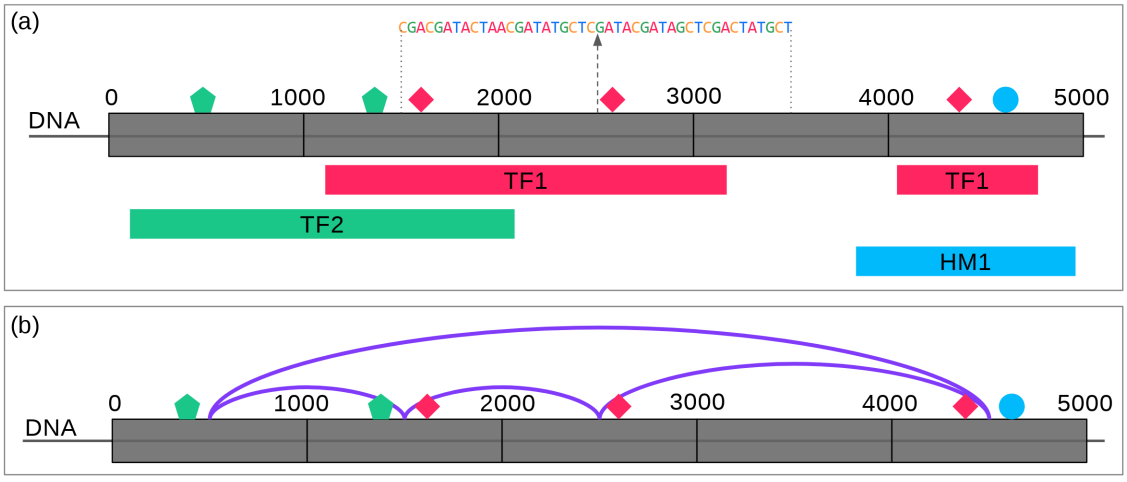

Figure 3.2: **(a) Sequence Data Processing.** We extract 2000bp sequences surround 1000bp windows for any window that has an overlapping ChIP-seq peak. **(b) 3D Genome Data Processing** we use Hi-C contacts between the 1000bp windows from [6] as edges in our 3D genome graph.

3D Genome Data. We then use 3D genome data from Hi-C contact maps to extract interaction evidence

between the DNA windows. We use 1000bp resolution intra-chromosome Hi-C data from [6] (for K562, the lowest resolution is 5000bp, so we upsample to get 1000bp resolution). We convert the Hi-C contact map for each chromosome into a graph whose nodes are 1000bp DNA windows and whose edges represent contact between two 1000bp windows. Since the full Hi-C contact for each chromosome is too dense, we rank each contact edge, and use the top 500,000 Hi-C contacts as edges per chromosome (each chromosome maps to a Hi-C graph). Contacts are ranked using the *SQRTVC* normalization from [6], which normalizes for the distance between two positions so that long-range contacts are included in the top 500k.

3.4 Method

Our goal is to learn a model f which takes in a DNA subsequence window $\mathbf{x}_i$ and predicts the probability of a set of chromatin labels $\hat{\mathbf{y}}_i = f(\mathbf{x}_i)$, where $\hat{\mathbf{y}}_i$ is an L dimensional vector. Our method, ChromeGCN uses three submodules for f: f_{CNN}, f_{GCN}, and f_{Pred}. The first module, f_{CNN}, models local sequence patterns from each window using a convolutional neural network. This module takes as input $\mathbf{x}_i$ and outputs a vector representation $\mathbf{h}_i = f_{CNN}(\mathbf{x}_i)$. The second module, f_{GCN}, models long range 3D genome dependencies between windows using a graph convolutional network. This module takes as input all window vectors $\mathbf{h}_i$ concatenated as $\mathbf{H}$, as well as their Hi-C relationships represented by adjacency matrix $\mathbf{A}$, and outputs refined representations of all windows $\mathbf{Z} = f_{GCN}(\mathbf{H}, \mathbf{A})$. The $\mathbf{z}_i$ of each window now encodes both window sequence patterns and the relationships between windows. We can then predict the chromatin labels using a classifier function on each $\mathbf{z}_i$ using $\hat{\mathbf{y}}_i = f_{Pred}(\mathbf{z}_i)$. An overview of ChromeGCN is shown in Fig. 3.1(c). The following subsections explain each submodule in detail.

3.4.1 Modeling Local Sequence Representations Using Convolutional Neural Networks

Following the recent successes in many chromatin label prediction tasks [43, 44, 59, 60, 7], we learn a representation of each genomic window sequence $\mathbf{x}_i$ using a convolutional neural network (CNN). CNNs have become the de facto standard for encoding short DNA windows due to their properties, which effectively capture local sequence structure. Each learned kernel, or filter, in CNNs effectively learns a DNA "motif", or short contiguous sequence representative of a particular output label [44]. Since many chromatin processes are hypothesized to be dependent on motifs [63], CNNs are a good choice for encoding DNA.

This module, f_{CNN}, takes an input genomic sequence window $\mathbf{x}_i$, and outputs an embedding representation vector $\mathbf{h}_i$. We represent window $\mathbf{x}_i$ of length τ as a one-hot encoded matrix $\mathbf{X}_i \in \mathbb{R}^{\tau \times n_{in}}$, where n_{in} is 4, representing the base-pair characters A,C,G, and T. Convolution with filters (i.e. learned motifs) of length $k < \tau$ takes an input data matrix $\mathbf{X}_i$ of size $\tau \times n_{in}$, and outputs a matrix $\mathbf{X'}_i$ of size $\tau \times n_{out}$, where n_{out} is the chosen dimension of the learned hidden representations:

$$\mathbf{X'}_{i\,t,u} = \sigma\left(\sum_{j=1}^{n_{in}}\sum_{z=1}^{k}\mathbf{W}_{u,j,z}\mathbf{X}_{i\,t+z-1,j}\right), \tag{3.1}$$

where $\mathbf{W} \in \mathbb{R}^{n_{out} \times n_{in} \times k}$ are the trainable weights, and σ is a function enforcing element-wise nonlinearity.

Eq. 3.1 can then be repeated for several layers where each successive layer uses a new $\mathbf{W}$ and n_{in} is replaced with n_{out} from the previous layer. In our implementation, we use six layers of convolution where each successive layer learns higher-order motifs of the window. After the convolutional layers, the output of the last layer is flattened into a vector and then linearly transformed into a lower-dimensional vector of size d, which we denote $\mathbf{h}_i$. Succinctly, the CNN module computes the following: $\mathbf{h}_i = f_{CNN}(\mathbf{x}_i)$ for each window.

3.4.2 Modeling Long-Range 3D Genome Relationships Using Graph Convolutional Networks

While CNN models work well on independent local window sequences, they disregard known long-range relationships between windows that are influential in the chromatin profile. One option would be to extend the window size. However, due to the 3D shape of DNA, long-range contact dependencies may be located millions of base-pairs apart, making current convolutional models infeasible. In this subsection, we introduce the f_{GCN} module, a method to explicitly and efficiently model such long-range interactions using graph convolutional networks.

Known long-range relationships in the 3D genome are available in the form of Hi-C contact maps. A Hi-C map can be represented as an adjacency matrix $\mathbf{A}$, where the nonzero elements indicate contacts in the 3D space between two DNA windows[1]. In our formulation, we represent sequence windows $\mathbf{x}_i$ as nodes on a graph, and $\mathbf{A}$ are the edges between the windows. We can then use a modified version of graph convolutional networks [16] (GCN) to update each $\mathbf{x}_i$ with its neighboring windows $\mathbf{x}_j, j \in \mathcal{N}(i)$, where $\mathcal{N}(i)$ denotes the neighbors of node i obtained from $\mathbf{A}$. The GCN works by revising a window's representation $\mathbf{h}_i^t$, where $\mathbf{h}_i^0$ is from the output of the first module, f_{CNN}. We denote t to represent the GCN layer index. Specifically, each

[1]In our experiments, we use a different adjacency matrix $\mathbf{A}$ for each chromosome (intra-chromosome Hi-C maps). However, we generalize a $\mathbf{A}$ to represent all possible window interactions (i.e., including inter-chromosome maps).

$\mathbf{h}_i^t$ is revised using a parameterized summation of neighbors, $\mathbf{h}_j^t, j \in \mathcal{N}(i)$:

$$\mathbf{h}_i^{t+1} = \sigma\left(\frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} \mathbf{h}_j^t \mathbf{W}^t\right), \tag{3.2}$$

where $\sigma(\cdot)$ is a non-linear activation function such as tanh and $\mathbf{W}^t \in \mathbb{R}^{d\times d}$ is a linear feature transformation matrix for the GCN layer t. Importantly, using the summation in Eq. 3.2, the representation of each DNA window $\mathbf{x}_i$ is updated based on the representation of its neighbors (windows that interact with $\mathbf{x}_i$ in the 3D genome). We can compute the simultaneous update of all windows together by concatenating all $\mathbf{h}_i$ denoted $\mathbf{H}^t \in \mathbb{R}^{N\times d}$ where N is the number of DNA windows and d is the dimension of each $\mathbf{h}_i$. The simultaneous update can then be written as:

$$\mathbf{H}^{t+1} = \sigma(\mathbf{A}'\mathbf{H}^t\mathbf{W}^t). \tag{3.3}$$

where $\mathbf{A}' = \hat{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\hat{\mathbf{D}}^{-\frac{1}{2}}$, is the normalized adjacency matrix and $\hat{\mathbf{D}}$ is the diagonal degree matrix of $(\mathbf{A}+\mathbf{I})$.

In our experiments, we use a variant of the graph convolutional network, which uses a gating function allowing the model to use or not use neighboring windows to update each $\mathbf{h}_i^t$:

$$\widetilde{\mathbf{H}}^t = \tanh\left(\mathbf{A}'\mathbf{H}^t\mathbf{W}_z^t\right) \tag{3.4}$$

$$\mathbf{g}^t = \text{sigmoid}\left(\widetilde{\mathbf{H}}^t\mathbf{w}_g^t\right) \tag{3.5}$$

$$\mathbf{H}^{t+1} = \text{diag}(\mathbf{g}^t)\widetilde{\mathbf{H}}^t + \text{diag}(\mathbf{1}-\mathbf{g}^t)\mathbf{H}^t \tag{3.6}$$

where $\mathbf{W}_z^t$ is a linear transformation matrix, $\mathbf{1}$ is a vector of all 1s, and $\mathbf{w}_g^t \in \mathbb{R}^d$ is used to compute the gating vector. Gating vector $\mathbf{g}^t$ allows the model to selectively choose between using the neighborhood representation of nodes, $\widetilde{\mathbf{H}}^t$, or the independent representation, $\mathbf{H}^t$. Equations 3.3-3.6 indicate one layer update of GCN window embeddings $\mathbf{H}$. In our experiments, we use a 2-layer gated GCN (i.e. t=2), and we denote the final output of all $\mathbf{H}^t$ as $\mathbf{Z}$, where vector $\mathbf{z}_i$ of $\mathbf{Z}$ represents the output of window i. In summary, the GCN module computes the following: $\mathbf{Z} = f_{GCN}(\mathbf{H}, \mathbf{A})$.

3.4.3 Predicting Label Probabilities for Each Window

After $\mathbf{Z}$ is computed, we then use a linear classifier layer, f_{Pred} to classify each $\mathbf{z}_i$ into its output space (a set of chromatin labels): $\hat{\mathbf{y}}_i = \sigma(\mathbf{z}_i^\top\mathbf{W}+\mathbf{b})$. In summary, the prediction $\hat{\mathbf{y}}_i$ for a particular input sample $\mathbf{x}_i$ can

be decomposed as three steps:

$$\hat{\mathbf{y}}_i = f_{Pred}(\mathbf{z}_i)$$
$$\mathbf{Z} = f_{GCN}(\mathbf{H}, \mathbf{A})$$
$$\mathbf{h}_i = f_{CNN}(\mathbf{x}_i).$$

3.4.4 Model Variations

To test the effectiveness of the long range dependencies, we use the following ChromeGCN variations. Each variation uses the same model, with different edge dependencies in the form of $\mathbf{A}$.

ChromeGCN$_{const}$. Instead of using Hi-C edges we use a constant set of nearby neighbors according to the 1D sequential DNA representation. We define each window $\mathbf{x}_i$'s neighbors to be the windows surrounding $\mathbf{x}_i$ (7 on each side: $\mathbf{x}_{i-7}, ..., \mathbf{x}_{i+7}$) which we denote as $\mathbf{A}_{const}$. $\mathbf{Z} = GCN(\mathbf{A}_{const}, \mathbf{H})$. This variant allows us to see whether the very long range interactions from the normalized Hi-C maps are useful.

ChromeGCN$_{Hi\text{-}C}$. This variation uses the original Hi-C adjacency matrix, $\mathbf{A}_{Hi\text{-}C}$. $\mathbf{Z} = GCN(\mathbf{A}_{Hi\text{-}C}, \mathbf{H})$. Hi-C contacts are close neighboring contacts. However, by using top 500k contacts after the $SQRT$ normalization for the Hi-C graph, we reduced some of this locality bias in the graph. This results in many of the edges being far away in the 1D space. This allows us to decouple the effects of local neighboring contacts (constant) and long-range (normalized Hi-C) contacts.

ChromeGCN$_{const+Hi\text{-}C}$. Lastly, we use a combination of the constant neighborhood around each window and the Hi-C adjacency matrix, which integrates close and far windows for each window. This varition uses the following function: $\mathbf{Z} = GCN(\mathbf{A}_{const+Hi\text{-}C}, \mathbf{H})$.

3.4.5 Model Details and Training

To circumvent GPU memory constraints of training end-to-end, we pretrain the f_{CNN} model by classifying each $\mathbf{h}_i$ with the classification function $\hat{\mathbf{y}}_i = f_{Pred}(\mathbf{h}_i)$. Once the pretraining converges on the training set, we use the trained weights $\mathbf{h}_i$ for each sample as fixed inputs to f_{GCN}. While we pretrain f_{CNN}, ChromeGCN is still end-to-end differentiable, making it possible to use sequence visualization methods such as DeepLIFT [70] for a particular window.

For all model predictions, we run the forward and the reverse complement through simultaneously and average the output of the two. All DNA window inputs are encoded using a lookup table that maps each

	GM12878			K562		
	Mean AUROC	Mean AUPR	Mean Recall at 50% FDR	Mean AUROC	Mean AUPR	Mean Recall at 50% FDR
CNN [7]	0.895	0.350	0.293	0.894	0.325	0.265
DanQ [59]	0.886	0.348	0.290	0.900	0.343	0.290
ChromeRNN	0.906	0.384	0.342	0.910	0.365	0.327
ChromeGCN$_{const}$	0.904	0.377	0.331	0.904	0.358	0.321
ChromeGCN$_{Hi\text{-}C}$	0.904	0.385	0.341	0.903	0.358	0.319
ChromeGCN$_{const+Hi\text{-}C}$	**0.909**	**0.395**	**0.356**	**0.912**	**0.372**	**0.338**

Table 3.2: Performance results. For both cell lines, GM12878 and K562, we show the average across all labels for three different metrics. Our method, using a graph convolutional network (GCN) to model long range dependencies helps improve performance over the baseline CNN model which assumes all DNA segments are independent.

character A, C, G, T, and N (unknown) to a d-dimensional vector. The output of the encoding is a $d \times \tau$ matrix, whereτ denotes sequence length ($\tau = 2000$ in our experiments).

All of our models are trained using stochastic gradient descent with momentum of 0.9 and a learning rate of 0.25. The CNN model is trained using a batch size of 64, and the GCN and RNN models are trained using an entire chromosome as a batch (since each is modeling the between window dependencies of an entire chromosome at once). The CNN model projects each window to a vector of dimension 128. The GCN uses two layers of feature dimension 128 at each layer.

ChromeGCN predicts the probabilities of all labels for each window: $\hat{\mathbf{y}}_i \in \mathbb{R}^L$, where L is the total number of labels. For our loss function, we use the mean binary cross-entropy across all samples N and labels L:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{M} \sum_{i=1}^{N} \frac{1}{L} \sum_{l=1}^{L} - \left(y_i^l \log \left(\hat{y}_i^l \right) + \left(1 - y_i^l \right) \log \left(1 - \hat{y}_i^l \right) \right) \tag{3.7}$$

3.5 Experiments and Results

3.5.1 Baselines

We compare against the state-of-the-art chromatin profile prediction model from [7], referred to as the CNN baseline, as well as the recurrent model from [59]. Since our model outputs labels for TFBS, HMs, and accessibility, motif-based methods [71] aren't applicable. Since we have 368,082 training samples, kernel-based methods such as [55] aren't applicable. [43] compared their CNN to a modified version of [55], which only used a small number of training samples, and the CNN model was significantly better. Our CNN baseline, [7] is an improved version from [43]. Furthermore, the focus of our study is to show that state-of-the-art deep learning models are missing important long range dependencies in the genome.

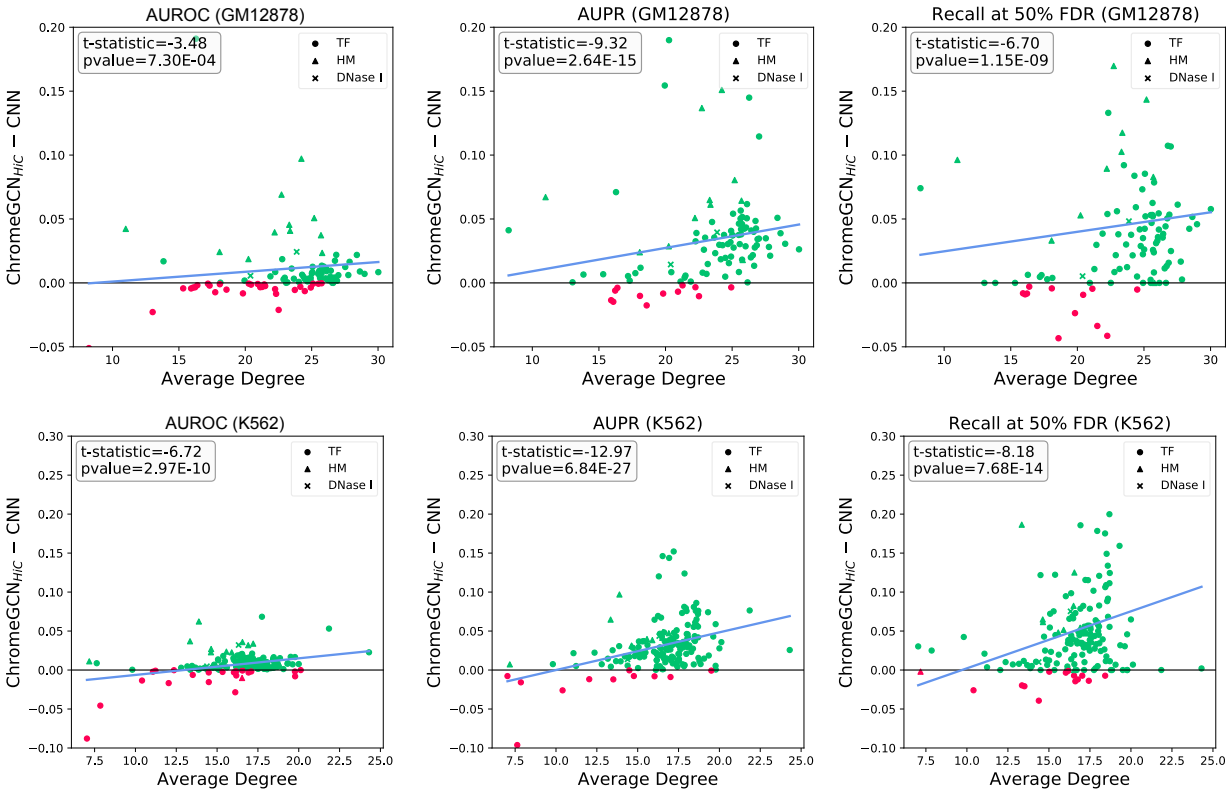

Figure 3.3: **Importance of incorporating long range interactions**. This figure shows a comparison of our ChromeGCN$_{Hi\text{-}C}$ method vs the baseline CNN [7] for 3 Metrics. Each point represents one chromatin profile label. The labels are sorted in the x-axis by the average degree of their positive windows. The y-axis indicates absolute increase of the ChromeGCN$_{Hi\text{-}C}$ over the CNN model. As the average degree increases, the improvement of the ChromeGCN$_{Hi\text{-}C}$ model increases over the CNN. Green points indicate ChromeGCN$_{Hi\text{-}C}$ performed better, red indicate the CNN performed better. The blue line shows the linear trend line. The ChromeGCN$_{Hi\text{-}C}$ is significantly better, as demsonstrated by the pvalues from a pairwise t-test.

CNN [7]. To illustrate the importance of the GCN, we compare against outputs from the f_{CNN} module: $\hat{\mathbf{y}}_i = f_{Pred}(\mathbf{h}_i)$. This is the 6-layer CNN model from [7] (we modify the last layer in order to extract a d dimensional feature vector output). This is the same CNN that is pretrained for ChromeGCN to produce each $\mathbf{h}_i$.

DanQ [59]. This model uses a recurrent neural network (RNN) on top of CNN outputs within a window. It still uses local sequence window inputs, but models relationships between sequence patterns via an LSTM.

ChromeRNN. As a baseline to compare against using GCNs for long range dependency modelling, we construct an RNN model on the window embeddings $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_N$. After pretraining the CNN module f_{CNN}, The RNN model takes in all window embeddings at once and models the sequential dependencies among windows: $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, ..., \mathbf{z}_N) = f_{RNN}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, ..., \mathbf{h}_N)$. As with ChromeGCN, the RNN is shared across

chromosomes, but does not cross chromosomes. In other words, the embeddings are updated one chromsome at a time $f_{RNN}(\mathbf{h}_i, \mathbf{h}_{i+1}, ..., \mathbf{h}_C)$ where C is the total number of windows for a chromosome. We note this is different from the DanQ baseline [59] which uses an RNN *within* windows [59]. ChromeRNN instead is for modeling dependencies *between* windows. In our experiments, we use an LSTM [49] with the same number of layers and hidden units as the GCN.

3.5.2 Prediction Performance

To evaluate the methods, we use area under the ROC curve (AUROC), area under the precision-recall curve (AUPR), and the Mean Recall at 50% False Discovery Rate (FDR) cutoff. Table 3.2 shows the mean metric results across all chromatin labels for each cell line. Modeling long-range dependencies results in significant improvements over the baseline CNN model, which does not account for such long range interactions. For instance, with respect to AUPR for GM12878, ChromeRNN improves upon the CNN from 0.350 to 0.384, and ChromeGCN outperforms ChromeRNN to achieve a mean AUPR of 0.395. Also, we see that the ChromeRNN outperforms DanQ, indicating that using an RNN to model between window dependencies is more important than within window features. Moreover, we can see that the ChromeGCN$_{Hi\text{-}C}$ models outperform ChromeRNN, indicating that not only the closely neighboring windows in the 1D space contribute to the improvements, but also the close neighbors in the 3D space, as indicated by the used Hi-C maps. ChromeGCN outperforms the baselines on the TF and DNase I labels, and ChromeRNN outperforms all other methods on the HM labels. This indicates that non-local modeling is particularly important for TF binding and accessibility. We provide the performance results of each label type (TF, HM, DNase I) are shown in Tables 3.3 and 3.4. We provide detailed plots of ROC and Precision-Recall curves in Figures 3.4 and 3.5.

Furthermore, since ChromeGCN models the window relationships explicitly and not recurrently, we obtain a significant speedup at test time over the ChromeRNN baseline. ChromeGCN achieves a 6.3x speedup on the three GM12878 test chromosomes, and 6.8x speedup at test time on the three K562 test chromosomes.

	TFs			HMs			DNase I		
	Mean AUROC	Mean AUPR	Mean Recall at 50% FDR	Mean AUROC	Mean AUPR	Mean Recall at 50% FDR	Mean AUROC	Mean AUPR	Mean Recall at 50% FDR
CNN [7]	0.909	0.328	0.262	0.796	0.478	0.469	0.801	0.657	0.750
DanQ [59]	0.899	0.325	0.260	0.794	0.479	0.463	0.792	0.650	0.723
ChromeRNN	0.914	0.354	0.299	**0.859**	**0.574**	**0.610**	**0.821**	0.689	**0.787**
ChromeGCN$_{const}$	0.913	0.348	0.291	0.849	0.554	0.58	0.815	0.68	0.777
ChromeGCN$_{Hi\text{-}C}$	0.916	0.362	0.309	0.819	0.516	0.528	0.814	0.683	0.774
ChromeGCN$_{const+Hi\text{-}C}$	**0.918**	**0.369**	**0.319**	0.845	0.552	0.584	0.820	**0.692**	0.783

Table 3.3: Performance on GM12878 for each label category

	TFs			HMs			DNase I		
	Mean AUROC	Mean AUPR	Mean Recall at 50% FDR	Mean AUROC	Mean AUPR	Mean Recall at 50% FDR	Mean AUROC	Mean AUPR	Mean Recall at 50% FDR
CNN [7]	0.905	0.314	0.256	0.78	0.410	0.317	0.792	0.625	0.683
DanQ [59]	0.909	0.331	0.279	0.796	0.438	0.365	0.797	0.638	0.693
ChromeRNN	0.917	0.349	0.305	**0.848**	**0.520**	**0.528**	0.817	0.664	0.742
ChromeGCN$_{const}$	0.911	0.342	0.301	0.84	0.507	0.507	0.810	0.657	0.729
ChromeGCN$_{Hi\text{-}C}$	0.912	0.346	0.306	0.807	0.454	0.412	0.812	0.657	0.730
ChromeGCN$_{const+Hi\text{-}C}$	**0.919**	**0.358**	**0.320**	0.837	0.495	0.484	**0.821**	**0.67**	**0.752**

Table 3.4: Performance on K562 for each label category

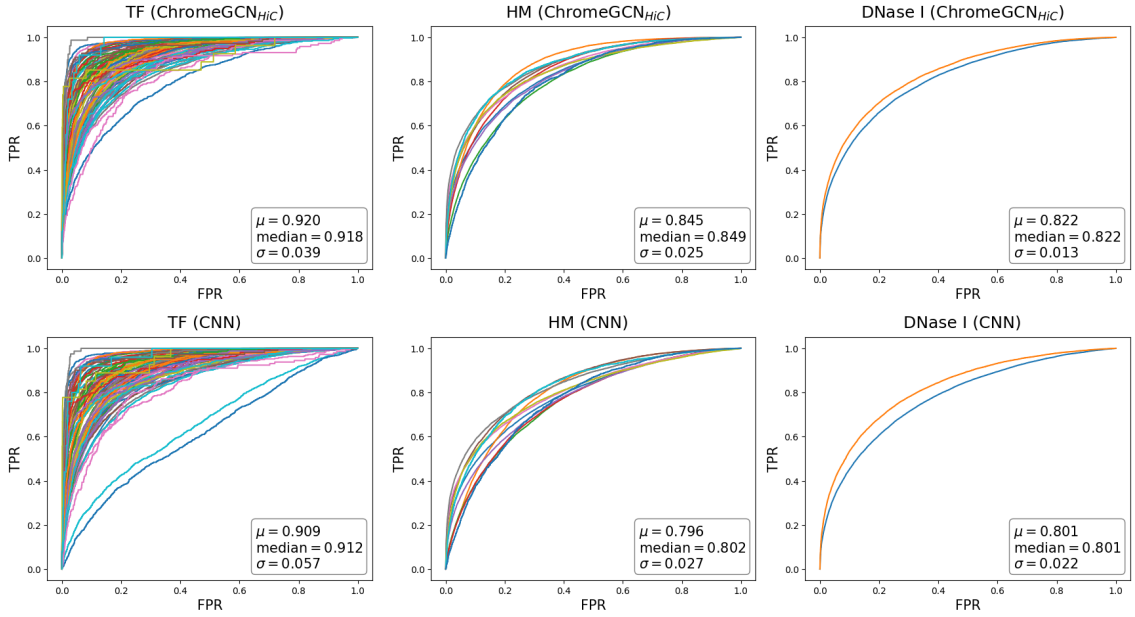

Figure 3.4: **ROC curves for GM12878** The top row shows the ChromeGCN$_{Hi\text{-}C}$ variant, and the bottom row shows the CNN[7]. The columns are divided into the 3 types of labels: transcription factors (TFs), histone modificiations (HMs), and DNA accesibility (DNase I). The color of each curve represents a different label, where they are consistent across columns. The box in each plot shows the statistics of the area under the curves (AUC). ChromeGCN outperforms the CNN for all Epigenetic state labels.

3.5.3 Analysis of Using Hi-C Data

Fig. 3.3 shows a a detailed comparison of ChromeGCN$_{Hi\text{-}C}$ vs the baseline CNN model across three different metrics for both GM12878 and K562. Each point represents a label, and the y-axis shows the absolute improvement of the ChromeGCN$_{Hi\text{-}C}$ model over the CNN. The labels are sorted on the x-axis by the average degree of the label's positive samples (i.e., windows where the label is positive) on the Hi-C map. We can see that for all three metrics, the improvements of the ChromeGCN over the CNN increase as the average degree of the labels increase. This indicates that the ChromeGCN is important for labels that have many neighbors in the Hi-C graph (i.e., those that are frequently in contact with other segments in the 3D space). Two of the transcription factors which obtain the highest performance increase (in the top 5) from using ChromeGCN

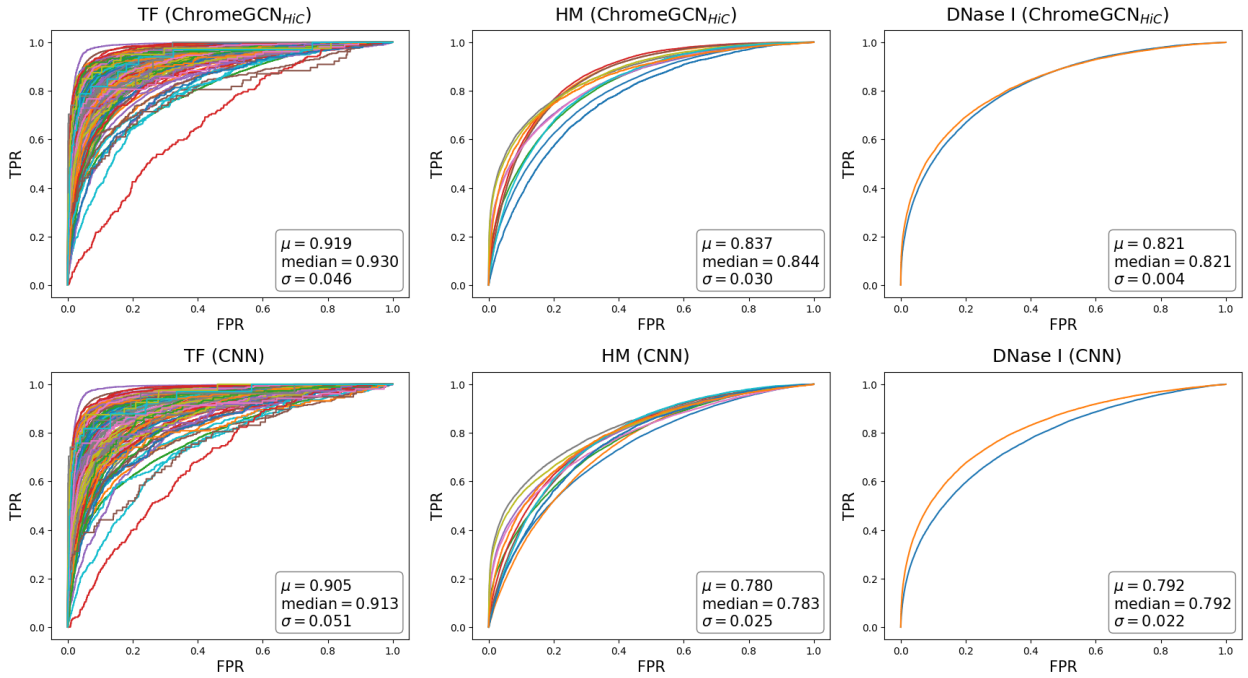

Figure 3.5: **ROC curves for K562** . The top row shows the ChromeGCN$_{Hi\text{-}C}$ variant, and the bottom row shows the CNN[7]. The columns are divided into the 3 types of labels: transcription factors (TFs), histone modificiations (HMs), and DNA accesibility (DNase I). The color of each curve represents a different label, where they are consistent across columns. The box in each plot shows the statistics of the area under the curves (AUC). ChromeGCN outperforms the CNN for all Epigenetic state labels.

over a CNN, CEBPB STAT3, are validated by [46], which show that these two TFs commonly co-occur with other TFs in the 3D space when binding.

The p-values shown are computed by a pairwise t-test across all labels. The ChromeGCN$_{Hi\text{-}C}$ model significantly outperforms the CNN model in all three metrics. Importantly, our results indicate that by using the long-range interactions given by Hi-C data, we can obtain improvements in modeling the chromatin profile labeling, resulting in better classification accuracy.

3.6 Visualizing and Understanding

While making accurate predictions is important in biomedical tasks, it is equally important to understand why models make their predictions. We are specifically interested in identifying input features that are responsible for a particular output value. Consequently, we aim to obtain a better understanding of why certain models work better than others, and investigate how they make their predictions by introducing several visualization techniques. To do so, we introduce two classes of genomic feature attribution methods:

identifying important long range interactions, and identifying important sequence (i.e. nucleotide) features.

3.6.1 Identifying Important Sequence Features and Local Interactions

In addition to Hi-C saliency maps which help us understand the important long range genomic interactions, we also want to understand sequence features that are correlated with output chromatin profile labels. To this end, we present the "Deep Motif Dashboard" (DeMo Dashboard), to understand the inner workings of deep neural network models for a genomic sequence classification task. The purpose of the DeMo Dashboard is to automatically discover and help us understand the underlying "motifs", or DNA sequence features that are important for a chromatin profile signal to occur. We do this by introducing a suite of different neural models and visualization strategies to see which ones perform the best and understand how they make their predictions. The proposed DeMo Dashboard allows us visualize and understand a model in three different ways: Saliency Maps, Temporal Output Scores, and Class Optimizations.

Sequence Saliency Maps

For a certain DNA sequence and a model's classification, a logical question may be: "which which parts of the sequence are most influential for the classification?" To do this, we seek to visualize the influence of each position (i.e. nucleotide) on the prediction. Our approach is similar to the methods used on images by Simonyan et al. et al. [72] and Baehrens et al. [73]. Given a sequence X_0 of length $|X_0|$, and class $c \in C$, a machine learning model provides a score function $S_c(X_0)$. We rank the nucleotides of X_0 based on their influence on the score $S_c(X_0)$. Since $S_c(X)$ is a highly non-linear function of X with deep neural nets, it is hard to directly see the influence of each nucleotide of X on S_c. Mathematically, around the point X_0, $S_c(X)$ can be approximated by a linear function by computing the first-order Taylor expansion:

$$S_c(X) \approx w^T X + b = \sum_{i=1}^{|X|} w_i x_i + b \tag{3.8}$$

where w is the derivative of S_c with respect to the sequence variable X at the point X_0:

$$w = \left.\frac{\partial S_c}{\partial X}\right|_{X_0} = \textit{saliency map} \tag{3.9}$$

This derivative is simply one step of backpropagation, and is therefore easy to compute. We do a pointwise multiplication of the saliency map with the one-hot encoded sequence to get the derivative values for the actual nucleotide characters of the sequence (A,T,C, or G) so we can see the influence of the character at each

position on the output score. Finally, we take the element-wise magnitude of the resulting derivative vector to visualize how important each character is regardless of derivative direction. We call the resulting vector a "saliency map[72]" because it tells us which nucleotides need to be changed the least in order to affect the class score the most. As we can see from equation 3.8, the saliency map is simply a weighted sum of the input nucleotides, where the each weight, w_i, indicates the influence of that nucleotide position on the output score.

Temporal Output Scores

Since DNA is sequential (i.e. can be read in a certain direction), it can be insightful to visualize the output scores at each timestep (position) of a sequence, which we call the temporal output scores. Here we assume an imaginary time direction running from left to right on a given sequence, so each position in the sequence is a timestep in such an imagined time coordinate. In other words, we check the RNN's prediction scores when we vary the input of the RNN. The input series is constructed by using subsequences of an input X running along the imaginary time coordinate, where the subsequences start from just the first nucleotide (position), and ends with the entire sequence X. This way we can see exactly where in the sequence the recurrent model changes its decision from negative to positive, or vice versa. Since our recurrent models are bi-directional, we also use the same technique on the reverse sequence. CNNs process the entire sequence at once, thus we can't view its output as a temporal sequence, so we use this visualization on just the RNN and CNN-RNN.

Class Optimization

The previous two visualization methods listed are representative of a specific testing sample (i.e. sequence-specific). Now we introduce an approach to extract a *class-specific* visualization for a model, where we attempt to find the best sequence which maximizes the probability of a positive class, which we call class optimization. Formally, we optimize the following equation where $S_+(X)$ is the probability (or score) of an input sequence X (matrix in our case) being a positive class computed by the softmax equation of our trained model for a specific chromatin profile label:

$$\arg\max_X S_+(X) + \lambda \|X\|_2^2 \tag{3.10}$$

where λ is the regularization parameter. We find a locally optimal X through stochastic gradient descent, where the optimization is with respect to the input sequence. In this optimization, the model weights remain unchanged. This is similar to the methods used in Simonyan et al. [72] to optimize toward a specific image

class. This visualization method depicts the notion of a positive class for a particular chromatin label and is not specific to any test sequence.

End-to-end Automatic Motif Extraction from the Dashboard

Our three proposed visualization techniques allow us to manually inspect how the models make their predictions. In order to automatically find patterns from the techniques, we also propose methods to extract motifs, or consensus subsequences that represent the positive binding sites. We extract motifs from each of our three visualization methods in the following ways: (1) From each positive test sequence we extract a motif from the saliency map by selecting the contiguous length-9 subsequence that achieves the highest sum of contiguous length-9 saliency map values. (2) For each positive test sequence, we extract a motif from the temporal output scores by selecting the length-9 subsequence that shows the strongest score change from negative to positive output score. (3) For each different TF, we can directly use the class-optimized sequence as a motif.

Experimental Setup

In order to evaluate our sequence visualization methods, we train and test three commonly used sequence-only (i.e. no Hi-C inputs) deep neural network models: convolutional neural network (CNN), recurrent neural network (RNN), and a combination of the two, a convolutional-recurrent neural network (CNN-RNN). Details of our models are included in the Appendix. We run our experiments on the 108 K562 cell ENCODE ChIP-Seq transcription factor (binary classification) datasets used in Alipanahi et al. [44]. Each TF dataset has an average of 30,819 training sequences (with an even positive/negative split), and each sequence consists of 101 DNA-base characters (A,C,G,T). Every dataset has 1,000 testing sequences (with an even positive/negative split). Positive sequences are extracted from the hg19 genome centered at the reported ChIP-Seq peak. Negative sequences are generated by dinucleotide-preserving shuffle of the positive sequences. Due to the separate train/test data for each TF, we train a separate model for each individual TF dataset.

Our CNN implementation involves a progression of convolution, nonlinearity, and maxpooling. This is represented as one convolutional layer in the network, and we test up to 4 layer deep CNNs. The final layer involves a maxpool across the entire temporal domain so that we have a fixed-size vector which can be fed into a softmax classifier. Figure 3.6 (a) shows our CNN model with two convolutional layers. The input one-hot encoded matrix is convolved with several filters (not shown) and fed through a ReLU nonlinearity to produce a matrix of convolution activations. We then perform a maxpool on the activation matrix. The output of the

first maxpool is fed through another convolution, ReLU, and maxpooled across the entire length resulting in a vector. This vector is then transposed and fed through a linear and softmax layer for classification.

Since there is no innate direction in genomic sequences, we use a bi-directional LSTM as our RNN model. In the bi-directional LSTM, the input sequence gets fed through two LSTM networks, one in each direction, and then the output vectors of each direction get concatenated together in the temporal direction and fed through a linear classifier. Figure 3.6 (b) shows our RNN model. The input one-hot encoded matrix is fed through an LSTM in both the forward and backward direction which each produce a matrix of column vectors representing the LSTM output embedding at each timestep. These vectors are then averaged to create one vector for each direction representing the LSTM output. The forward and backward output vectors are then concatenated and fed to the softmax for classification.

Considering convolutional networks are designed to extract motifs, and recurrent networks are designed to extract temporal features, we implement a combination of the two in order to find temporal patterns between the motifs. Given an input matrix $\mathbf{X} \in \mathbb{R}^{T \times n_{in}}$, the output of the CNN is $\mathbf{Z} \in \mathbb{R}^{T \times n_{out}}$. Each column vector of $\mathbf{Z}$ gets fed into the RNN one at a time in the same way that the one-hot encoded vectors get input to the regular RNN model. The resulting output of the RNN $\mathbf{H} \in \mathbb{R}^{T \times d}$, where d is the LSTM embedding size, is then averaged across the temporal domain (in the same way as the regular RNN), and fed to a softmax classifier. Figure 3.6 (c) shows our CNN-RNN model. The input one-hot encoded matrix is fed through one layer of convolution to produce a convolution activation matrix. This matrix is then input to the LSTM, as done in the regular RNN model from the original one-hot matrix. The output of the LSTM is averaged, concatenated, and fed to the softmax, similar to the RNN.

Understanding Neural Networks Using the DeMo Dashboard

To evaluate the dashboard visualization methods, we first manually inspect the dashboard visualizations to look for interpretable signals. Figure 3.7 shows examples of the DeMo Dashboard for three different TFs and positive TFBS sequences. We apply the visualizations on the best performing models of each of the three architectures. Each dashboard snapshot is for a specific TF and contains (1) JASPAR[74] motifs for that TF, which are the "gold standard" motifs generated by biomedical researchers, (2) the positive TFBS class-optimized sequence for each architecture (for the given TF of interest), (3) the positive TFBS test sequence of interest, where the JASPAR motifs in the test sequences are highlighted using a pink box, (4) the saliency map from each model on the test sequence, and (5) forward and backward temporal output scores from the recurrent architectures on the test sequence. In the saliency maps, the more red a position is, the

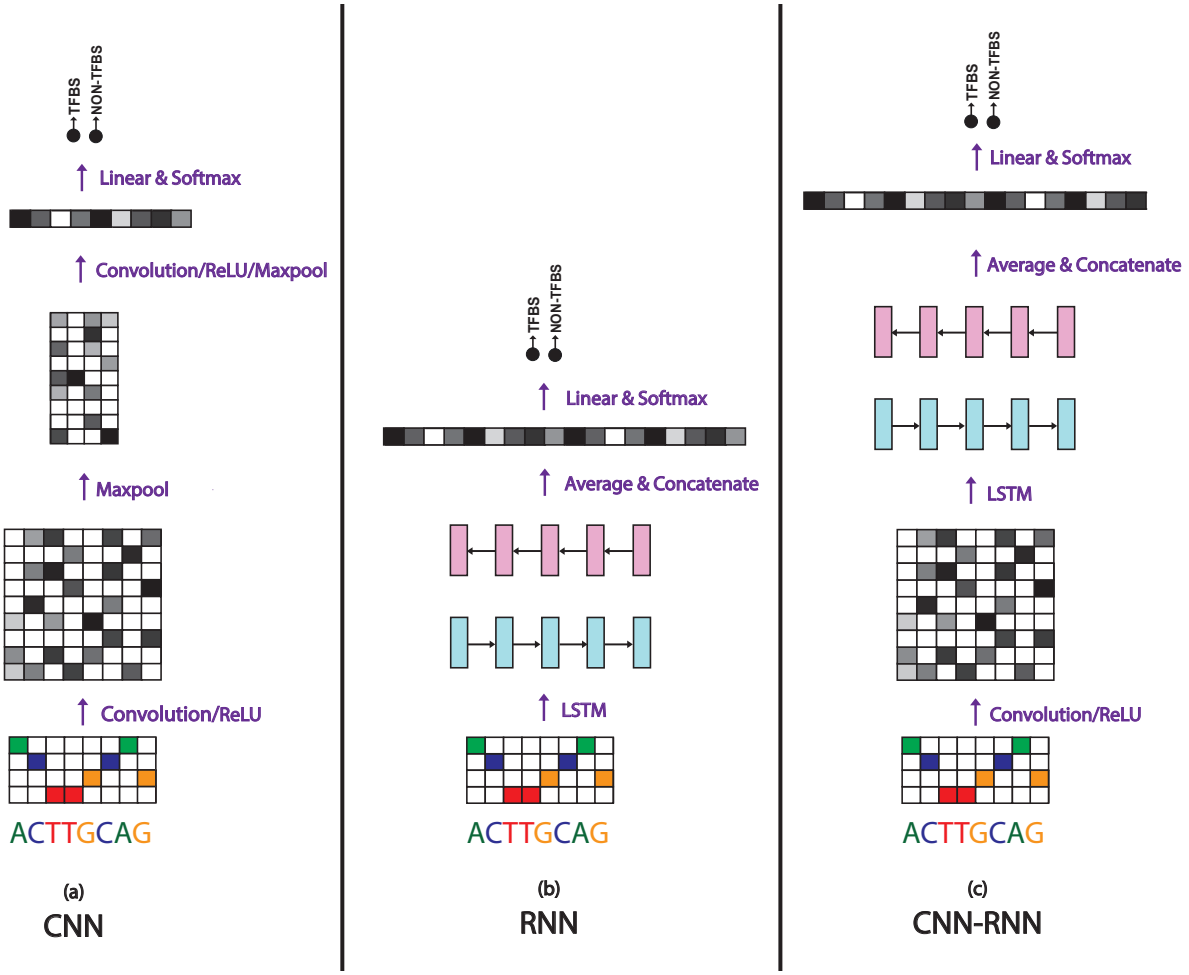

Figure 3.6: **Deep Motif Dashboard Local Sequence Model Architectures.** Each model has the same input (one-hot encoded matrix of the raw nucleotide inputs), and the same output (softmax classifier to make a binary prediction). The architectures differ by the middle "module", which are **(a)** Convolutional, **(b)** Recurrent, and **(c)** Convolutional-Recurrent.

Figure 3.7: **Deep Motif Dashboard**. Dashboard examples for GATA1, MAFK, and NFYB positive TFBS Sequences. The top section of the dashboard contains the Class Optimization (which does not pertain to a specific test sequence, but rather the class in general). The middle section contains the Saliency Maps for a specific positive test sequence, and the bottom section contains the temporal Output Scores for the same positive test sequence used in the saliency map. The very top contains known JASPAR motifs, which are highlighted by pink boxes in the test sequences if they contain motifs.

	Saliency Map (out of 500)	Conv Activations [44] (out of 500)	Temporal Output (out of 500)	Class Optimization (out of 57)
CNN	243.9	173.4	N/A	19
RNN	138.6	N/A	53.5	11
CNN-RNN	168.1	74.2	113.2	13

Table 3.5: JASPAR motif matches against DeMo Dashboard and baseline motif finding methods using Tomtom.

more influential it is for the prediction. In the temporal outputs, blue indicates a negative TFBS prediction while red indicates positive. The saliency map and temporal output visualizations are from the same positive test sequence. The numbers next to the model names in the saliency map section indicate the score outputs of that model on the specified test sequence.

Saliency Maps (middle section of dashboard). By visual inspection, we can see from the saliency maps that CNNs tend to focus on short contiguous subsequences when predicting positive bindings. In other words, CNNs clearly model "motifs" that are the most influential for prediction. The saliency maps of RNNs tend to be spread out more across the entire sequence, indicating that they focus on all nucleotides together, and infer relationships among them. The CNN-RNNs have strong saliency map values around motifs, but we can also see that there are other nucleotides further away from the motifs that are influential for the model's prediction. For example, the CNN-RNN model is 99% confident in its GATA1 TFBS prediction, but the prediction is also influenced by nucleotides outside the motif. In the MAFK saliency maps, we can see that the CNN-RNN and RNN focus on a very wide range of nucleotides to make their predictions, and the RNN doesn't even focus on the known JASPAR motif to make its high confidence prediction.

Temporal Output Scores (bottom section of dashboard). For most of the sequences that we tested, the positions that trigger the model to switch from a negative TFBS prediction to positive are near the JASPAR motifs. We did not observe clear differences between the forward and backward temporal output patterns.

In certain cases, it's interesting to look at the temporal output scores and saliency maps together. An important case study from our examples is the NFYB example, where the CNN and RNN perform poorly, but the CNN-RNN makes the correct prediction. We observe that the CNN-RNN is able to switch its classification from negative to positive, while the RNN never does. To understand why this may have happened, we can see from the saliency maps that the CNN-RNN focuses on two distinct regions, one of which is where it flips its classification from negative to positive. However, the RNN doesn't focus on either of the same areas, and may be the reason why it's never able to classify it as a positive sequence. The fact that the CNN is not able to classify it as a positive sequence, but focuses on the same regions as the CNN-RNN (from the

saliency map), may indicate that it is the temporal (i.e. long range) dependencies between these regions which influence the binding. In addition, the fact that there is no clear JASPAR motif in this sequence may show that the traditional motif approach is not always the best way to model TFBSs.

Class Optimization (top section of dashboard). Class optimization on the CNN model generates concise representations which often resemble the known motifs for that particular TF. For the recurrent models, the TFBS positive optimizations are less clear, though some aspects stand out (like "AT" followed by "TC" in the GATA1 TF for the CNN-RNN). We notice that for certain models, their class optimized sequences optimize the reverse complement motif (e.g. NFYB CNN optimization). The class optimizations can be useful for getting a general idea of what triggers a positive TFBS for a certain TF.

Automatic Motif Extraction from Dashboard. In order to evaluate each model's capability to automatically extract motifs, we compare the found motifs of each method (introduced in section 3.6.1) to the corresponding JASPAR motif, for the TF of interest. We do the comparison using the Tomtom [75] tool, which searches a query motif against a given motif database (and their reverse complements), and returns significant matches ranked by p-value indicating motif-motif similarity. Table 3.5 summarizes the motif matching results comparing visualization-derived motifs against known motifs in the JASPAR database. We are limited to a comparison of 57 out of our 108 TF datasets by the TFs which JASPAR has motifs for. We compare four visualization approaches: Saliency Map, Convolution Activation[44, 76], Temporal Output Scores and Class Optimizations. The first three techniques are sequence specific, therefore we report the average number of motif matches out of 500 positive sequences (then averaged across 57 TF datasets). The last technique is for a particular TFBS positive class.

We can see from Table 3.5 that across multiple visualization techniques, the CNN finds motifs the best, followed by the CNN-RNN and the RNN. However, since CNNs perform worse than CNN-RNNs by AUC scores, we hypothesize that this demonstrates that it is also important to model sequential interactions among motifs. In the CNN-RNN combination, CNN acts like a "motif finder" and the RNN finds dependencies among motifs. This analysis shows that visualizing the model's classifications can lead to a better understanding of using neural networks for chromatin profile prediction.

3.6.2 Identifying Important Long Range Interactions

One benefit of the ChromeGCN formulation is that we use explicit long-range window dependencies for epigenetic state prediction. As a result, we propose a method to identify the important dependencies for

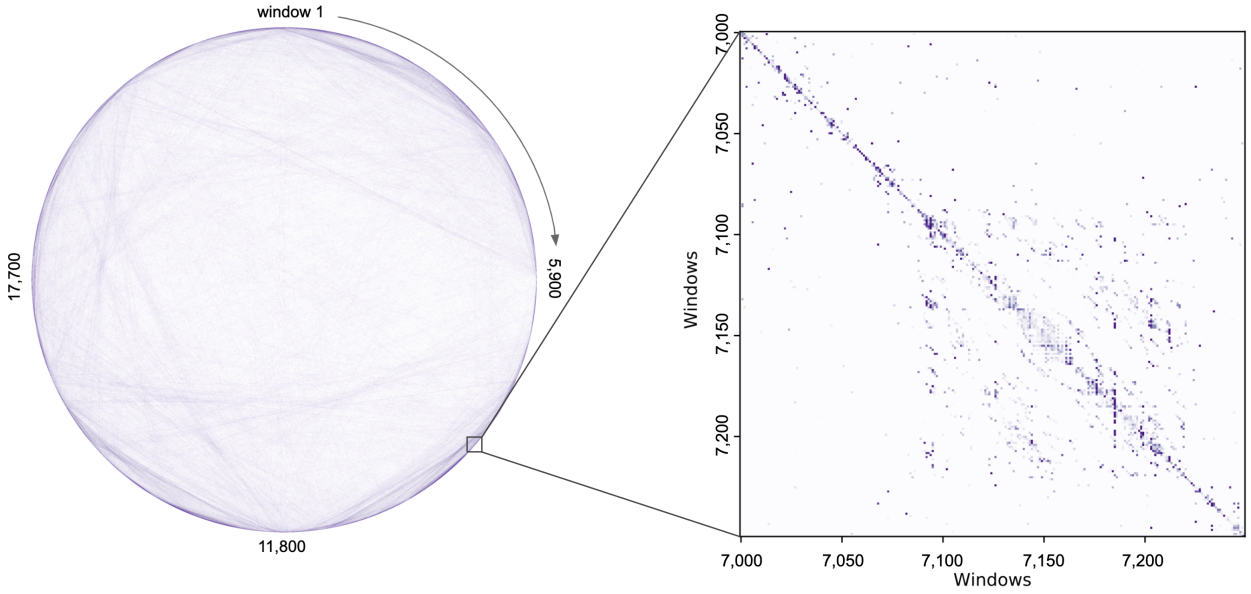

Figure 3.8: **Hi-C Saliency Map Visualization.** *Left:* Saliency Map for all 500k edges in $\mathbf{A}_{Hi\text{-}C}$ for GM12878 Chromosome 8 (total of 23,600 windows). The darker the line, the more important that edge was for predicting the correct chromatin profile, indicating that the Hi-C data was used by the GNN for that particular interaction. *Right:* Fine grained analysis of the Chromosome 8 Saliency Map. This figure shows the normalized Saliency Map values for for 250 windows (total of 250kbp input) in chromosome 8.

ChromeGCN's predictions. We call our proposed method Hi-C saliency maps. Saliency maps were introduced by [72] to understand the importance of each pixel in an input $\mathbf{x}_i$ for the prediction of the image's true class. We instead are trying to understand the importance of each edge in $\mathbf{A}'$ for the prediction of the epigenetic state over all windows. The $\mathbf{A}'$ saliency map is defined as the absolute value gradient of a true class prediction $\hat{y}_i^\ell$ with respect to $\mathbf{A}'$, where ℓ is a true class for sample i. The absolute value gradient is then element-wise multiplied by $\mathbf{A}'$ to zero out "non-edges". Since there are N samples, and there can be multiple true labels ℓ for a particular sample $\hat{\mathbf{y}}_\mathbf{i}$, we define the Hi-C saliency map, $S_{Hi\text{-}C}$, as the accumulated absolute gradient over all samples and true labels w.r.t $\mathbf{A}'$:

$$S_{Hi\text{-}C} = \sum_{i=1}^{N} \sum_{\ell \in \hat{\mathbf{y}}_\mathbf{i}} \mathbf{A}' \circ \left| \frac{\partial \hat{y}_i^\ell}{\partial \mathbf{A}'} \right|, \tag{3.11}$$

where $\circ$ is the Hadamard product. Since the saliency map of all windows N are accumulated, we normalize $S_{Hi\text{-}C}$ across each row to a 0-1 range so that we can interpret the edges at each window.

While Hi-C contact maps tell us *where* the contacts are, Hi-C saliency maps show us *how important* each contact is for the epigenetic states. We define Eq. 3.11 to be over all labels, but we can easily visualize the Hi-C saliency, or important edges for one particular label. We show both the full Hi-C saliency across all labels, as well as for one specific label (YY1) in the experiments.

Long Range Interaction Visualization

A significant merit of ChromeGCN is that by using known 3D genome relationships, we can find and visualize the critical relationships for epigenetic state prediction. To understand how important the Hi-C edges are for the predictions of ChromeGCN, we visualize the saliency map of $\mathbf{A}'$, as explained in Section 3.6.2.

Fig. 3.8 shows the Hi-C saliency map for chromosome 8 in GM12878. Fig. 3.8 (left) shows all 500k Hi-C contacts used chromosome 8. Windows are represented as points along the circle, with a total of 23,600 windows. Lines between the windows represent Hi-C edges, and the darkness of the line represents the saliency, or importance of that edge for chromatin state prediction across all windows in chromosome 8. Fig. 3.8 (right) shows the saliency map for 250 windows (total of 250kbp input) in chromosome 8. Cell (i, j) tells us the importance of window column j for the prediction of window row i labels.

While Fig 3.8 shows the Hi-C saliency map for all epigenetic state labels, we can also visualize the Hi-C saliency map for individual labels. The inner loop of Eq. 3.11 changes to only use the label of interest.

3.7 Discussion and Extensions

We've focused mostly on predicting the chromatin profile from DNA. But ultimately we're interested in predicting how and when genes are expressed or repressed. There are a lot of opportunities to extend this work by incorporating gene expression as well as looking at the causal relationships between the sequence and gene expression profiles. Several works have shown that it's possible to accurately predict gene expression from chromatin profile signals [77, 78]. However, we want to predict expression directly from sequence as this allows us to analyze how mutations can effect all downstream processes. By including gene expression in our set of output labels, we can both predict the effects of mutations on gene expression, as well as understand the interactions between sequence, chromatin profile, and expression.

3.8 Summary

In this chapter, we present ChromeGCN, a novel framework that combines both local sequence and long-range 3D genome data (via Hi-C data) for chromatin profile prediction. We show that Graph neural network models can effectively exploit and discover important interactions in DNA for functional genomics. We demonstrate experimentally that ChromeGCN outperforms previous state-of-the-art methods that only use local sequence data. Additionally, we introduce several methods to identify and visualize important genomic features and

interactions. These include Hi-C saliency maps to uncover important long range interactions, as well as the Deep Motif Dashboard to identify important sequence features.

While we demonstrate the importance of ChromeGCN on the task of chromatin profile prediction, ChromeGCN is a generic model for incorporating 3D genome structure into any genome sequence prediction task. ChromeGCN introduces an effective and efficient framework to model such relationships for better chromatin modeling, as well as an easy way to interpret important relationships.

Chapter 4

Modeling Virus-Host Protein-Protein Interactions with DeepVHPPI

Proteins are essential biomolecules that carry out critical functions within our bodies. A key property of proteins is that they interact with other proteins in order to carry out specific functions. So in order to determine what a protein really does, we must first know who it interacts with. This is critically important especially when studying viruses because viruses infect humans through protein-protein interactions. For example the Covid-19 Spike protein interacts with the Human ACE2 protein to enter human cells and replicate. Predicting and understanding protein-protein interactions directly from protein sequences is is critical for preventing and slowing the emergence of novel infectious diseases. The problem with sequence-based interaction models is that they do not naturally incorporate important structural and semantic information of individual proteins that are important for predicting interactions.

Research Question 2: *Can we incorporate individual protein structure and semantics into a sequence based protein-protein interaction prediction model?*

In this chapter, we propose DeepVHPPI, a novel deep learning framework combining a self-attention-based transformer architecture and a transfer learning training strategy to predict interactions between human proteins and virus proteins that have novel sequence patterns. We show how to use this framework to predict novel interactions for SARS-CoV-2, H1N1, and Ebola. Furthermore, we demonstrate how our framework can be used to predict and understand how virus mutations can influence interactions, and therefore infections.

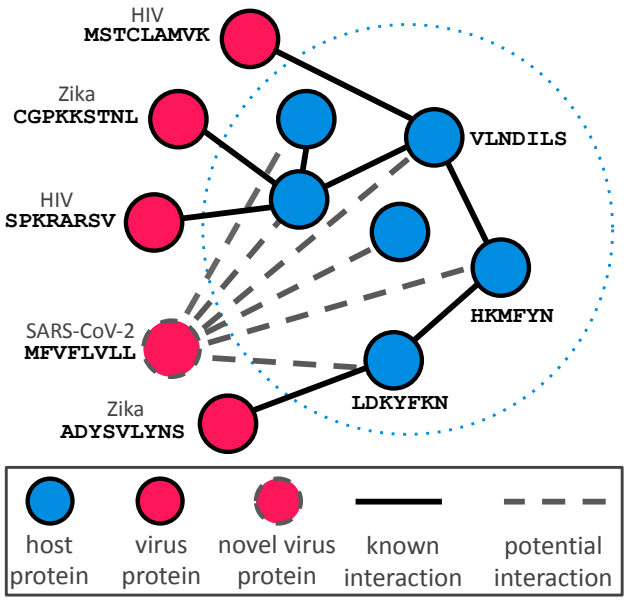

Figure 4.1: Virus-Host Protein-Protein Interactions (PPI). Overview of our task, where there is a set of previously known protein-protein interactions. Our goal is to predict all possible Virus–Human interactions for a novel virus protein, such as SARS-CoV-2.

4.1 Introduction

A protein-protein interaction (PPI) denotes a critical process where two proteins come in contact with each other to carry out specific biological functions. Virus proteins, such as those from the 2019 novel coronavirus, also known as SARS-CoV-2, interact with human proteins to infect the human body, and ultimately overtake physiological functions (e.g., alveolar gas exchange). Accordingly, protein-protein interactions are often the subject of intense research by virologists and pharmaceutical scientists. Knowing and understanding which host proteins a virus with a novel sequence pattern may interact with is crucial. Such discoveries will expedite our understanding of virus mechanisms and may aid in the development of vaccines, diagnostics, therapeutics, and antibodies.

We aim to infer possible interactions between all host proteins and a novel virus protein or a novel variant. This setup is beneficial for three reasons. First, our model can predict an initial set of interactions if experiments have not yet been done. Second, our model can expand the initial set of experimental interactions, resulting in a more complete interactome. Finally, such computational models enable us to test hypotheses such as the effect of mutations.

While protein-protein interaction information is expensive to obtain, protein sequence information is cheap and fast. in this chapter, we propose a deep neural network (DNN) based framework, DeepVHPPI,

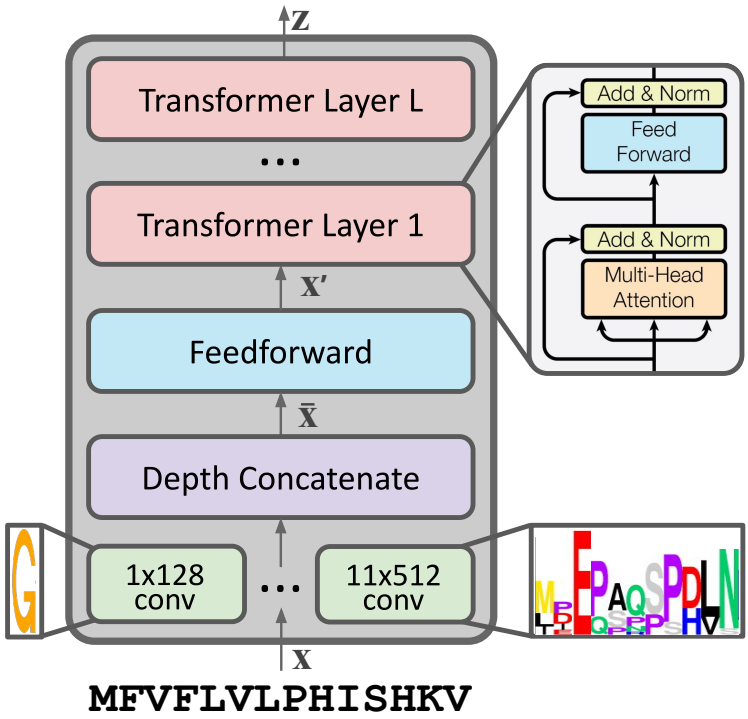

Figure 4.2: DeepVHPPI Architecture. A one-hot encoded sequence $\mathbf{x}$ gets input to the convolutional layers to find protein "motifs". The convolution outputs are then concatenated along the depth dimension and input to a feedforward layer. Finally, several Transformer encoder layers model the dependencies between the learned convolutional motifs, producing a final representation $\mathbf{z}$. The representation can then be used for any arbitrary classifier layer to predict protein properties.

to predict protein interactions between virus proteins and host proteins using sequence information alone. DeepVHPPI includes two key designs: (1) Motivated by the evidence that co-occurring short polypeptide sequences between interacting protein partners appear to be conserved across different organisms [79], we introduce a novel DNN architecture to learn short sequence patterns, or "protein motifs" via self-attention based deep representation learning. (2) since virus-host PPI data is limited, we propose a transfer learning approach to pretrain the network on general protein syntax and structure prediction tasks. The objective of this transfer learning approach is to improve generalization on the task of predicting protein-protein interactions involving novel virus proteins with unseen sequences.

In summary, we make the following contributions:

- We introduce the DeepVHPPI, a novel deep neural framework for protein sequence based Virus–Host PPI prediction for novel virus proteins or virus proteins with novel variants.
- DeepVHPPI combines a self-attention based transformer architecture and transfer learning for PPI prediction in the context of novel virus sequences (where no previous interactions are known).

- We evaluate DeepVHPPI with validated interactions on Virus–Host PPIs across three virus types: SARS-CoV-2, H1N1 and Ebola datasets. We show that DeepVHPPI outperforms the previous state-of-the-art methods, as well as provide an analysis of SARS-CoV-2 Spike protein mutations.

4.2 Background and Task Formulation

Proteins are biomolecules that are comprised of a linear chain of amino acids. This allows them to be described by a sequence of tokens (each token is one amino acid (AA)). The dictionary of possible tokens contains 20 standard AAs, two non-standard AAs: selenocysteine and pyrrolysine, two ambiguous AAs, and a special character for unknown (i.e. missing) AAs. In other words, we can represent proteins as strings built from a dictionary V of size $|V| = 25$. We represent a protein $\mathbf{x}$ as a sequence of characters $x_1, x_2, ..., x_L$. Each character x_i is one possible amino acid from V. Proteins rarely act in isolation but instead interact with other proteins to perform many biological processes. This is referred to as a protein-protein interaction (PPI).

Task Formulation. Viruses infect a host through Virus-Host PPIs. Therefore, predicting which host proteins a virus protein will bind to is a key step in understanding viral pathogenesis[1] and designing viral therapies. Identifying virus-host PPI interactions can be formulated as a binary classification problem: "given virus protein sequence $\mathbf{x}^q$ and host protein sequence $\mathbf{x}^k$, does the pair interact or not?". Fig. 4.1 gives a visual representation of the types of PPIs that we consider with our model. that occur within the human body. There are three types of proteins in this diagram: Host (human) proteins, previously known virus proteins, and a novel virus protein. As shown with solid lines, there are a set of known interactions between a pairs of host proteins as well as between known virus and host proteins. We consider known host-host interactions, known virus-host interactions, and unknown virus-host interactions. Fig. 4.1 visualizes the case of predicting unknown interactions between human proteins and SARS-CoV-2 proteins, given what is known about interactions with proteins from HIV and Zika. Our target task is to predict all possible unknown sets of interactions between the novel virus and host proteins, as shown with a dashed line. This formulation motivates the use of transfer learning because we want to transfer the learned interactions from known viruses to a novel virus. Specifically, we're concerned here with proteins from novel viruses, which is different than a novel protein from a known virus in our training set

Biological Experiments to Detect PPIs. It remains difficult to accurately uncover the full set of protein-protein interactions from biological experiments. Traditionally, PPIs have been studied individually

[1]Mechanisms by which virus infection leads to disease in the target host

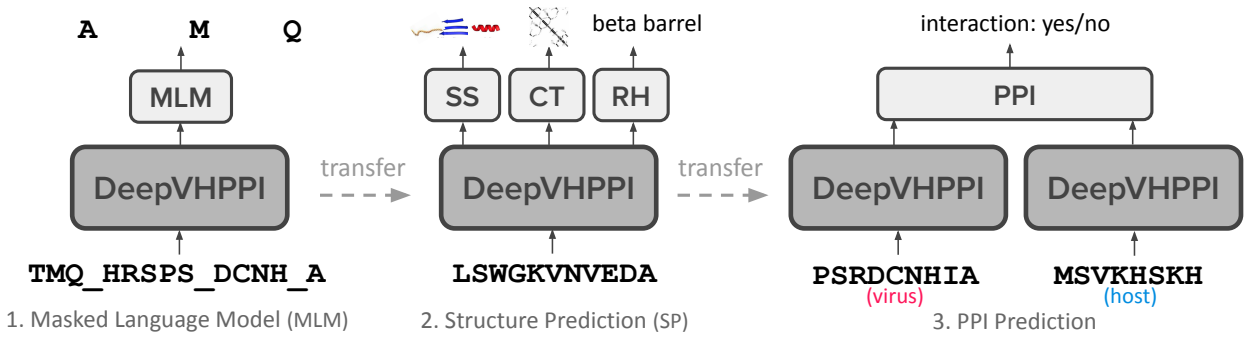

Figure 4.3: Transfer Learning Framework for DeepVHPPI. First, we pretrain the network on the Masked Language Model (MLM) task from a large repository of unlabeled protein sequences. Second, we further pretrain the network on a set of Structure Prediction (SP) tasks including secondary structure (SS), contact (CT), and remote homology (RH). Finally, we fine-tune the network on the protein-protein interaction (PPI) prediction task. The base DeepVHPPI shown as the large dark grey block is shared across all tasks, and each task uses its own classifier, shown as small light grey blocks.

through the use of genetic, biochemical, and biophysical techniques such as measuring natural affinity of binding partners in-vivo or in-vitro [80]. While accurate, these small-scale experiments are not suitable for full proteome analyses [81]. This is because, for example, there are roughly $|P_h| \approx 20,000$ different human proteins, and $|P_v| \approx 26$ different virus proteins (in SARS-CoV-2, not considering mutated variants), so the potential search space of V–H interactions is $|P_v| \times |P_h| =$ 0.5M. This number can grow significantly larger when you consider virus protein variants.

High-throughput technologies, such as yeast two-hybrid screens (Y2H) [82] and Affinity-purification–mass spectrometry (AP-MS) [83, 84], are chiefly responsible for the relatively large amount of PPI evidence. Notably, the first experimental study for SARS-CoV-2 interactions used AP-MS [84]. However, these datasets are often incomplete, noisy, and hard to reproduce [85]. The resulting low sensitivity of high-throughput experiments is unfavorable when trying to fully understand how the virus interacts with humans.

Past Machine Learning based PPI Prediction Studies. Most previous computational methods to predict PPIs have focused on within-species interactions [86, 87, 88, 89, 79, 90, 91, 92, 93]. These methods do not easily generalize to cross-species interactions (e.g., V–H) [94]. Few methods have attempted to predict cross-species protein interactions between humans and a novel virus [2, 94]. Furthermore, previous methods operating at the sequence level do not use structural information from previously known proteins to aid learning [4]. By training virus–host interactions from a variety of viruses and leveraging prior structural information, our proposed model, DeepVHPPI, allows us to predict the host interactions of an unseen virus protein.

When designing machine learning models to predict V–H PPIs, two challenges stand out: (1) Existing

Dataset	Category	Output Shape	|Total|	|Train|	|Valid|	|Test|
Swiss-Prot	MLM	$L \times \|V\|$	562,280	562,280	N/A	N/A
Secondary Structure	SP	$L \times 3$	11,361	8,678	2,170	513
Contact	SP	$L \times L$	25,563	25,299	224	40
Homology	SP	1195×1	13,766	12,312	736	718
SARS-CoV-2	PPI	1×1	815,279	199,346	49,836	610,950
H1N1 [2]	PPI	1×1	22,291	21,910	N/A	381
Ebola [2]	PPI	1×1	22,982	22,682	N/A	300

Table 4.1: Datasets: For each category of training: Language Model (LM), Intermediate (SP) and PPI, we provide the dataset output type and training/validation/test set sizes. L represents the sequence length, and $|V|$ represents the vocabulary size.

Method	**SS**	**Contact**	**Homology**
One-hot [95]	0.69	0.29	0.09
Alignment [95]	**0.80**	0.64	0.09
ResNet [95]	0.70	0.20	0.10
LSTM [95]	0.71	0.19	0.12
Transformer [95]	0.70	0.32	0.09
DeepVHPPI	0.70	0.51	0.12
DeepVHPPI + MLM	0.71	0.58	0.22
DeepVHPPI (multi-task)	0.64	0.53	0.13
DeepVHPPI + MLM (multi-task)	0.71	**0.70**	**0.38**

Table 4.2: Structure prediction (SP) pretraining task results. For SS and Homology, accuracy is reported. For Contact, precision at $L/5$ for for medium and long-range contacts is reported.

sequence analysis tools focus on global alignment patterns while PPIs mostly depend on local binding motif patterns. (2) It is especially difficult for virus proteins that are unknown or are new variants, since there is limited or no experimental interaction data for those sequences. This requires the machine learning model to transfer knowledge from one domain (previously known sequences) to a new domain (novel virus sequences). We argue that this is a realistic task when an unknown virus is newly discovered. In other words, we want to rapidly predict all the host interactions of a newly sequenced virus protein. In this work, we propose a deep learning based pipeline to combine neural representation learning and transfer learning for solving the listed obstacles. Recent literature shows some successful transferability of large scale deep learning models on protein sequences to multiple downstream tasks [95, 5]. To the authors' best knowledge, we are the first to adapt the self-attention based transfer learning to the virus-host PPI prediction task.

4.3 Proposed DNN Framework for Virus Host PPI Prediction: DeepVHPPI

We denote each amino acid in a protein sequence as $\mathbf{x}_{\text{CLS}}, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$, where $\mathbf{x}_i \in \mathbb{R}^{|V|}$ denotes a one-hot vector and $\mathbf{x}_{\text{CLS}}$ is a special classification token. Given virus protein $\mathbf{x}^a \in \mathbb{R}^{n \times |V|}$ and human protein $\mathbf{x}^b \in \mathbb{R}^{n \times |V|}$, the goal of DeepVHPPI is to predict the interaction likelihood $\hat{y}$ of the pair of proteins. In

this section, we explain the DeepVHPPI architecture, as shown in Fig. 4.2, which maps a protein sequence $\mathbf{x} \in \mathbb{R}^{n \times |V|}$ to representation $\mathbf{z} \in \mathbb{R}^{n \times d}$. In section 4.3.1, we introduce the Transformer module which maps a protein sequence $\mathbf{x}$ to a hidden representation, $\mathbf{z}$. In section 4.3.2, we introduce the classification module which takes as input both the virus protein hidden representation, $\mathbf{z}^a$, and the host protein hidden representation, $\mathbf{z}^b$, and outputs the likelihood that the two proteins interact.

4.3.1 Transformer Layers to Learn Representations of Protein Sequences

Transformers [19] have obtained state-of-the-art results in many domains such as natural language [50], images [96], and protein sequences [5]. A Transformer encoder layer is a parameterized function mapping input token sequence $\mathbf{x} \in \mathbb{R}^{n \times d}$ to $\mathbf{z} \in \mathbb{R}^{n \times d}$. At a high level, a Transformer encoder layer "transforms" the representations of input tokens (e.g., amino acids) by modeling dependencies between them in the form of attention. The importance, or weight, of token $\boldsymbol{x}_j$ with respect to $\boldsymbol{x}_i$ is learned through attention. Each Transformer encoder layer performs the following computation on input $\mathbf{x}$:

$$\alpha_{i,j}^{(h)} = \text{softmax}_j \left(\frac{\left\langle Q^{(h)}\left(\mathbf{x}_i\right), K^{(h)}\left(\mathbf{x}_j\right) \right\rangle}{\sqrt{k}} \right) \tag{4.1}$$

$$\mathbf{u}_i' = \sum_{h=1}^{H} \mathbf{W}_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j} V^{(h)}\left(\mathbf{x}_j\right), \tag{4.2}$$

$$\mathbf{u}_i = \text{LayerNorm}\left(\mathbf{x}_i + \mathbf{u}_i'\right), \tag{4.3}$$

$$\mathbf{z}_i' = \mathbf{W}_2^T \text{ GELU}\left(\mathbf{W}_1^T \mathbf{u}_i\right) \tag{4.4}$$

$$\mathbf{z}_i = \text{LayerNorm}\left(\mathbf{u}_i + \mathbf{z}_i'\right), \tag{4.5}$$

where $Q^{(h)}\left(\mathbf{x}_i\right)$=$\mathbf{W}_{h,q}^T \mathbf{x}_i$, $K^{(h)}\left(\mathbf{x}_i\right)$=$\mathbf{W}_{h,k}^T \mathbf{x}_i$, $V^{(h)}\left(\mathbf{x}_i\right)$=$\mathbf{W}_{h,v}^T \mathbf{x}_i$, and $\mathbf{W}_{h,q}, \mathbf{W}_{h,k}, \mathbf{W}_{h,v} \in \mathbb{R}^{d \times k}$, $\mathbf{W}_1 \in \mathbb{R}^{d \times m}, \mathbf{W}_2 \in \mathbb{R}^{m \times d}$, $\mathbf{W}_{c,h} \in \mathbb{R}^{k \times d}$. H, k, m, and d are hyperparameters where H is the total number of Transformer "heads", and k, m, and d are weight dimensions. GELU is a nonlinear layer [97], and LayerNorm is Layer Normalization [28]. The final $\mathbf{z}$ representation after L layers is the output of the Transformer encoder, which can then be used by a classification layer. We note that Transformers are invariant to the sequence length, always producing an output of the same length as the input.

Convolutional Layers to Extract Local Motif Patterns. Protein sequences have short, local patterns known as sequence motifs, that have been a major bioinformatics tool for years [98]. If we view amino acids

as the protein analog of natural language characters, motifs are analogous to words. In particular, virus proteins that successfully mimic host proteins and interact with other host proteins often display similar motifs to the target of mimicry [99]. To take advantage of these patterns, we introduce an architecture variant that stacks convolutional layers and transformer layers. The key contribution of this variation is to automatically learn sequence motifs via convolutional layers (motif module), and compose local patterns together via deeper transformer layers. Our motif module utilizes different length convolutional filters to find motifs directly from sequence end-to-end.

Specifically, we apply six temporal convolutional filters of sizes {(1×128), (3×256), (5×384), (7×512), (9×512), (11×512)} to the one-hot encoded protein sequence $\boldsymbol{x} \in \mathbb{R}^{\in L \times |V|}$, where the first number of each filter is the width and the second number is the depth. Each filter is zero-padded to preserve the original sequence length. We depth-concatenate the output of the convolutional, producing $\bar{\boldsymbol{x}} \in \mathbb{R}^{\in L \times 2304}$. $\bar{\boldsymbol{x}}$ is fed to a Feedforward layer to produce a $L \times d$ matrix $\boldsymbol{x}' = \mathbf{W}_1^T \,\text{GELU}\left(\mathbf{W}_2^T \bar{\boldsymbol{x}}\right)$. Finally, to encode positional information we add sinusoidal position tokens [19] to the $\boldsymbol{x}'$ matrix. This output $\boldsymbol{x}'$ is used as input to a Transformer encoder.

Using several convolutional filters of varying size allows the model to learn a diverse set of motifs. Specifically, in our implementation, the set of filters allows the model to learn 2304 unique motifs of varying lengths. DeepVHPPI is illustrated in Fig. 4.2 (left).

4.3.2 Classification Layer to Predict Protein-Protein Interactions

The Convolutional and Transformer layers map virus protein sequence $\mathbf{x}^a$ to $\mathbf{z}^a$ and host sequence $\mathbf{x}^b$ to $\mathbf{z}^b$. The final layer of DeepVHPPI is to predict the interaction likelihood of $\mathbf{x}^a$ and $\mathbf{x}^b$. We first obtain a single vector representation of each protein using the the classification token outputs from the Transformer, $\mathbf{z}^a_{\text{CLS}}$ and $\mathbf{z}^b_{\text{CLS}}$. In other words, each protein is fed into the shared Transformer model, and outputs an independent vector. Using these representations, we can predict $\hat{y}$, the likelihood that the two proteins interact with one another:

$$\mathbf{v} = \text{concat}(\mathbf{z}^a_{\text{CLS}}, \mathbf{z}^b_{\text{CLS}}) \tag{4.6}$$

$$\hat{y} = \mathbf{w}_2\big(\,\text{GELU}\,(\mathbf{W}_1\mathbf{v} + b)\,\big), \tag{4.7}$$

where $\mathbf{W}_1 \in \mathbb{R}^{2d \times d}$, and $\mathbf{w}_2 \in \mathbb{R}^{1 \times d}$ are projection matrices, and $\mathbf{v} \in \mathbb{R}^{2d \times 1}$ is a concatenation of the two protein representations. GELU is a nonlinear activation layer [97]. The virus sequence is always the first d

weights in the concatenated representation, so the classifier is not invariant to the protein ordering. $\hat{y}$ is then fed through a sigmoid function to obtain the interaction probability.

4.3.3 Proposed Training: Transfer Learning for Virus–Host Protein-Protein Interaction Prediction

Our proposed DeepVHPPI network allows us to predict the interaction likelihood of two proteins given only sequence information of each protein. With this framework, we are faced with several difficulties in order to predict Virus-Host interactions. First, there is limited Virus-Host PPI data available to train on. In particular, there are few or no interactions known for novel virus protein sequences. Second, protein structure information is important for accurate PPI prediction [100] Using sequence features alone may not be sufficient for predicting certain interactions, but obtaining structure for novel proteins is a slow process. Both of these obstacles require a model that can generalize from knowledge learned in related protein prediction tasks. To this end, we introduce a "transfer learning" three-step training procedure. This involves pretraining the Convolutional and Transformer layers (Section 4.3.1) of DeepVHPPI, before fine-tuning the entire network (Section 4.3.1 and 4.3.2) on the PPI task.

The first step is to pretrain the DeepVHPPI using Masked Language Model (MLM) pretraining in order to learn generic representations from unlabeled protein sequences; the second step is to further pretrain the network using Structure Prediction (SP) to learn 3D structural representations; and the third step is to finetune the network on the Virus–Host PPI data for previously known viruses. Pretraining the network allows it to learn representations that transfer well to the PPI task for novel (i.e., unseen) virus sequence. An overview of our proposed training procedure is shown in Figure 4.3, and we explain each training step below. Each task uses a task-specific classifier, shown as MLM, SS, CT, RH, and PPI in Figure 4.3. In other words, the DeepVHPPI is shared between all tasks, but the classifier layers are not.

MLM: Masked Language Model Pretraining

Recent literature on learning self-supervised representations of natural language have shown that pretraining using self-supervised and supervised methods encourages the model to learn semantics about the input domain that can help prediction accuracy on new tasks [24, 101, 102, 103, 50]. In order to learn basic protein semantics and syntax, leveraging large databases of protein sequences is paramount. Masked language model (MLM) training is a self-supervised technique to allow a model to build rich representations of sequences.

Specifically, given a sequence $\mathbf{x}$, the MLM objective optimizes the following loss function:

$$\mathcal{L}_{\text{MLM}} = \mathbb{E}_{\mathbf{x}\sim X}\mathbb{E}_M \sum_{i\in M} -\log p\left(\mathbf{x}_i \mid \mathbf{x}_{/M}\right) \tag{4.8}$$

where M is a set of indices to mask, replacing the true character with a dummy mask character. The total loss is a sum of each masked character's negative log likelihood of the true amino acid $\mathbf{x}_i$ its context set of characters $\mathbf{x}/M$. Specifically, for each training sample, we mask out a random 15% of the token positions for prediction. If the i-th token is chosen, it is replaced with (1) the *MASK* token 80% of the time (2) a random token 10% of the time (3) the unchanged i-th token 10% of the time. We use the output of the Transformer encoder, $\mathbf{z}$ to predict the likelihood of each amino acid a each missing token $\mathbf{x}_i$ using the following linear mapping:

$$\hat{\mathbf{y}}_i = \mathbf{W}\mathbf{z}_i + b, \tag{4.9}$$

with learned matrix $\mathbf{W} \in \mathbb{R}^{|V|\times d}$, bias b, and DeepVHPPI output $\mathbf{z}_i \in \mathbb{R}^{d\times 1}$. $\hat{y}_i$ is then fed through a softmax function to obtain class probabilities, $p\left(x_i \mid x_{/M}\right)$.

SP: Structure Prediction Pretraining

Masked language model pretraining uses large amounts of unlabeled data to learn protein sequence semantics. However, a key aspect of whether two proteins interact or not is the structure of each protein. Obtaining structural information is slow and expensive, so we generally do not have structure information for all proteins from a novel virus. We leverage existing structural information to further pretrain DeepVHPPI and learn structure from sequence by predicting known structures. We consider three structure-based classification tasks: (1) Secondary Structure (SS) prediction (2) Contact prediction, and, (3) Homology prediction. Each task is explained below.

Secondary Structure Prediction. Protein secondary structure is the three dimensional form of local segments of proteins. Each character in the sequence can be labeled by its secondary structure, which is one of $|C|$ classes where $C = \{\text{Helix, Strand, Other}\}$. This results in a sequence tagging task where each input amino acid character $\mathbf{x}_i$ is mapped to a label $y_i \in C$. We predict the likelihood of each class for x_i using the following linear mapping:

$$\hat{\mathbf{y}}_i = \mathbf{W}\mathbf{z}_i + b, \tag{4.10}$$

with learned matrix $\mathbf{W} \in \mathbb{R}^{|C|\times d}$, bias b, and DeepVHPPI output $\mathbf{z}_i \in \mathbb{R}^{d\times 1}$. $\hat{y}$ is then fed through a softmax function to obtain class probabilities.

Contact Prediction. Protein contact maps are a simplified depiction of the global 3D structure protein, where binary contact points indicated interactions in the 3D space. Contact prediction aims to predict the contact of each set of amino acid pairs in the sequence. Pair $(x_i\ ,\ x_j)$ of input amino acids from sequence $\mathbf{x}$ is mapped to a label $y_{ij} \in \{0, 1\}$ indicating whether or not the amino acids are physically close (< 8Å apart) to each other. To produce the contact likelihood of pair (x_i, x_j), we use the following formula which preserves non-directionality of contacts:

$$\hat{y} = \big((\mathbf{z}_i\mathbf{W}_1 \cdot \mathbf{z}_j\mathbf{W}_2) + (\mathbf{z}_i\mathbf{W}_2 \cdot \mathbf{z}_j\mathbf{W}_1)\big)/2, \tag{4.11}$$

where $\{\mathbf{W}_1, \mathbf{W}_2\} \in \mathbb{R}^{d\times d}$. $\hat{y}$ is then fed through a sigmoid function to obtain the contact probability.

Remote Homology Detection. Remote homologues are pairs of proteins that share the same functional class, but have drastically different sequences. The goal of remote homology detection is predict the structural and functional class of a protein. Since proteins evolve, many proteins are structurally (and thus, functionally) similar, although their sequences are slightly different. Accurately predicting the homology of a protein would allow the model to group similar structural proteins together. We consider predicting the remote homology of a protein in terms of structural "fold" classes (e.g. Beta Barrel). This is a protein classification task where each input sequence $\mathbf{x}$ is mapped to a label $y \in C$, where $|C| = 1195$ different possible protein folds. We use the designated *CLS* token from the DeepVHPPI to predict one of the $|C|$ labels for a given sequence. We use a single linear layer mapping $\mathbf{z}_i$ to a $|C|$-dimensional vector:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{z}_i + b, \tag{4.12}$$

where $\mathbf{W} \in \mathbb{R}^{|C|\times d}$ is a projection matrix and $\mathbf{z}_i \in \mathbb{R}^{d\times 1}$ is the *CLS* token output vector from the Transformer. $\hat{y}$ is fed through a softmax function to obtain class probabilities.

Multi-task Training. For secondary structure and contact prediction, we use a binary cross-entropy loss, and for remote homoloy, we use cross entropy. We train all three structure prediction tasks simultaneously by sampling a new task uniformly in each gradient descent batch.

Finetuning on Virus–Host Protein-Protein Interaction Task.

After MLM and SP pretraining, the final task is to finetune the model using the classifier in Section 4.3.2 on the known (i.e. experimentally validated) PPI data for previously known virus proteins. Once this model is trained, we can use it on pairs of Virus–Host interaction sequences where the virus was not used during

Method	AUROC	AUPR	F1(%)	P@100
Embedding+RF [94]	0.748	0.071	**0.116**	0.126
DeepVHPPI	0.740	0.065	0.104	0.129
DeepVHPPI+MLM	0.751	0.070	0.110	0.147
DeepVHPPI+MLM+SP	**0.753**	**0.076**	0.114	**0.151**

Table 4.3: Human and SARS-CoV-2 Interaction Predictions. Each metric is reported as the mean across all virus proteins. Best results are reported in bold.

	H1N1			Ebola		
Method	AUROC	AUPR	F1(%)	AUROC	AUPR	F1(%)
SVM [2]	0.886	-	76.2	0.867	-	76.0
DeepVHPPI	0.903	0.898	84.4	0.912	0.953	86.0
DeepVHPPI+MLM	0.908	0.903	85.5	0.953	0.961	**90.4**
DeepVHPPI+MLM+SP	**0.945**	**0.948**	**86.5**	**0.968**	**0.974**	89.6

Table 4.4: Virus–Human PPI Tasks from Zhou et al. [2]. Best results are in bold. "-" indicates the metric was not reported.

training. The pretraining method on generic tasks learn structures of proteins which generalize well to unseen proteins.

Method	AUROC	AUPR	F1
SVM [2]	0.858	-	79.2
Embedding + RF [94]	0.871	-	79.8
DeepVHPPI+MLM+SP	**0.886**	**0.806**	**80.6**

Table 4.5: Virus–Human PPI Tasks from Barman et al. [3]. Best results are in bold. "-" indicates the metric was not reported.

Method	AUROC	AUPR	F1
DeNovo [4]	-	-	81.9
SVM [2]	0.897	-	84.2
DeepVHPPI+MLM+SP	**0.989**	**0.991**	**95.9**

Table 4.6: SLiM PPI Tasks from Eid et al. [4]. Best results are in bold. "-" indicates the metric was not reported.

4.4 Related Work

Protein-Protein Interaction Prediction. Many previous PPI works focus on developing intra-species interactions [104, 86, 105, 106, 107, 108, 109]. In other words, they would have one model for only Human–Human interactions and another model for only Yeast–Yeast interactions. Cross-species interaction prediction instead relates to where each protein in the interaction comes from a different species. Many works predict cross-species PPIs where the testing set contains proteins that are in the training set [110, 111, 112, 113, 3]. These methods do not reflect the real-world setting for a novel virus since we don't have training proteins available for the virus. Additionally, PPI prediction methods generally perform much better for test pairs that share components with a training set than for those that do not [114]. Few works have focused on the more difficult task of cross-species interaction prediction where one of the protein species is completely unseen during training, which is what our work tackles. DeNovo [4] used an SVM for cross species interaction

prediction. Yang et al. [94] introduce a deep learning embedding method combined with a random forest. Zhou et al. [2] improved DeNovo's SVM for novel Virus–Human interaction.

Protein Sequence Classification. Machine learning methods have achieved considerable results predicting properties of proteins that have yet to be experimentally validated by experimental studies. [115, 116] introduce multitask deep learning models for sequence labeling tasks such as SS prediction. [5, 95, 117] focus on methods of language model pretraining for generalizable representations of sequences. In particular, [95, 5] and [118] showed that self-supervised pretraining can produce protein representations that generalize across protein domains.

Transformers. Transformers [19] obtained state-of-the-art results on several NLP tasks [50]. One problem with the vanilla Transformer model on token level inputs is that locality is not preserved. [119] used varying convolutional filters on characters at the word level and took the mean of the output to get a single vector representation for each word. Since proteins have no inherent "words" we use the convolutional output for each character as its local word. Instead of using character level inputs, word or byte-pair encodings can be used in order to preserve the local structure of words in text [120].

Transfer Learning. Our work relates to several others in natural language processing where pretraining is used to transfer knowledge from both unlabeled and related labeled datasets [121, 122, 123]. Transfer learning is closely tied with few-shot learning [124, 125], which typically aims to use representations from prior tasks to generalize. Transformers are particularly well-fitted for transfer learning as their parallelizable architecture allows for fast pretraining on large datasets [50, 126]. It has been shown that this large-scale pretraining generalizes well enough for accurate few-shot learning [127].

4.5 Experimental Setup and Results

4.5.1 Model Details and Evaluation Metrics

DeepVHPPI Variations and Details We evaluate three variants of our model: (1) DeepVHPPI: this is the base model which uses no pretraining, only training on the target PPI task. (2) DeepVHPPI+MLM: this variant uses the language model pretraining and finetuning on the PPI task. (3) DeepVHPPI+MLM+SP: this uses both language model pretraining and supervised structure/family prediction pretraining before finetuning on the PPI task. We test both single task and multi-task models on the 3 SP tasks. We use the

multi-task trained model for all PPI tasks. We train all models using a 12-layer transformer of size d=712 with H=8 attention heads.

For all training and testing, we clip protein sequences to 1024 length. For language model pretraining, we use a batch size of 1024, a linear warmup, and max learning rate of 1e-3. For all other tasks, we use a batch size of 16 with max learning rate of 1e-5. The language model is trained for 60 epochs, and all others are trained for 100. All models are trained with an Adam optimizer [128] and 10% dropout. Our models are implemented in PyTorch and we run each model on 4 NVIDIA Titan X GPUs. Masked language model pretraining (MLM) takes approximately 3 days, structure prediction (SP) pretraining takes 1 day, and the PPI task takes 3 days. Testing on ∼0.5M PPI pairs takes about 1 hour.

Metrics. For the supervised pretraining tasks, we use the metrics reported by previous work [95]. For the PPI task, we are largely focused on ranking interaction predictions based on probability, so we report two non-thresholding metrics: area under the ROC curve (AUROC), and area under the precision-recall curve (AUPR). We additionally report F1 scores where we consider thresholds [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]. As done in previous works, the results are selected on the best performing test epoch for each metric. For the SARS-CoV-2 dataset, we evaluate each metric for each virus protein individually since we are interested in the accuracy of predicting human interactions for specific virus proteins. The reported results are the mean value across all 25 virus proteins. For this dataset, we also report precision at 100 (P@100).

4.5.2 Pretraining Tasks (MLM and SP)

Datasets. For the masked language model (MLM) task, we train DeepVHPPI on all Swiss-Prot protein sequences [129]. Swiss-Prot is a collection of 562,253 manually reviewed, non-redundant protein sequences from 9,594 organisms. It includes most human and known virus proteins, allowing our model to learn the distribution of both types. We then further pretrain the model with structure pretraining (SP) tasks. For secondary structure prediction, the original data is from [130] and we report accuracy. For contact prediction, data is from [131]. We report precision of the $L/5$ most likely contacts (where L is the sequence length) for medium and long-range contacts. For remote homology prediction, the data is from [132], and we report accuracy. In all 3 tasks, we use the train/validation/test splits from [95].

Baselines. We compare our model against three deep learning methods: a vanilla Transformer, an LSTM [49], and ResNet [133], all run by [95]. Our DeepVHPPI uses the same Transformer model size (number of trainable parameters) as the vanilla transformer, and similar model size to the LSTM and ResNet. We do not compare to the pretrained models from [95] since we use a different pretraining dataset. We also compare

our method against two baseline methods from [95]. "One-hot" uses one-hot feature inputs that are fed to simple classifiers such as an MLP or 2-layer ConvNet. "Alignment" uses sequence alignment features (BLAST or HHBlits), which are matrices that encode evolutionary information about the protein [134, 135].

Results. Here we investigate the benefits of the DeepVHPPI on the structure prediction tasks. Table 4.2 shows results on the three SP datasets. Our proposed DeepVHPPI performs as well or better than baseline methods, aside from alignment methods on SS prediction. Self supervised language modeling adds improvement over the base DeepVHPPI. Multi-task training with language model pretraining outperforms all other non-alignment methods. We do not evaluate the performance of the MLM task itself since better MLM performance does not always translate to better downstream task performance [5].

4.5.3 SARS-CoV-2–Human PPI Task

Dataset. While there may be no known Virus–Human interactions for a novel virus, there are many experimentally validated interactions for previous viruses. Our proposed approach is to train an interaction model on known V–H interactions (as indicated by solid lines in Fig 4.1), and then test on all possible V–H interactions for the novel virus proteins (as indicated by dotted lines in Fig 4.1). We explain our full training and testing setup below. A summary of the datasets used is provided in Table 4.1.

Training Data. We use the V–H dataset from Yang et al. [136], which is based on data from the Host-Pathogen interaction Database (HPIDB; version 3.0) [137]. Note this only contains Host-Pathogen interactions, and not contain Host-Host interactions (i.e. only connections between red and blue proteins in Fig. 4.1). This dataset processed interactions from large-scale mass spectrometry experiments, resulting in 22,653 experimentally verified host-virus PPIs as a positive sample set. The authors chose negative pairs based on the 'Dissimilarity-Based Negative Sampling' [4]. The selected ratio of positive to negative samples is 1:10. Following Yang et al., we use the same training set (80%) and an independent validation set (20%) for model training and hyperparameter selection, respectively.

Testing Data. For the novel virus testing dataset, we use the 13,947 known SARS-CoV-2–Human interactions from the BioGRID database (Coronavirus version 4.1.190) [138]. All BioGRID interactions are experimentally validated, many from [84]. Considering all 20,365 SwissProt human proteins, we label all other pairs from the total space of 20,365*26 to be non-interacting (a total of 529,490). It is important to note that the labeled "negative" samples contain many pairs of proteins that interact but are not known to do so. This can result in an overestimation of the false positive rate [89].

Baselines. We compare our method to the pretrained Embedding+RF method from Yang et al. [94], which uses Doc2Vec to embed protein sequences and then a random forest to classify pairs. To the best of our knowledge, no other methods provide code or a browser service to run novel protein interactions.

Results. Table 4.3 shows our results for SARS-CoV-2. Across most metrics, our method DeepVHPPI outperforms the baseline method. MLM and SP pretraining help generalize to our target PPI task, where the non-pretrained models aren't as accurate. We note that the testing dataset is highly imbalanced. In other words, most interactions (99.7%) are negative, or non-interacting. Thus, the AUROC metric is not indicative of good results. We turn our attention to the AUPR metrics, where our method performs the best. This confirms our hypothesis that pretraining on large protein datasets learn evolutionary structures of proteins which generalize well to unseen proteins. This is a promising result not only for SARS-CoV-2, but for potential future novel viruses.

4.5.4 Other Virus–Host PPI Tasks (H1N1 and Ebola)

Datasets. In our SARS-CoV-2 dataset, we explain a testing scenario where we have no knowledge of the true V–H interactions, resulting in a large possible interaction space (all possible $|P_v| \times |P_h|$ interactions). Zhou et al. [2] created V–H datasets where they hand selected the negative interactions based on the known positives, making sure that they had an even positive/negative split. While this setup is unrealistic for a true novel virus (because we don't know which ones are positive), we compare to their results to show the strength of our method. Zhou et a. provide two H–V datasets, H1N1 and Ebola.

In the H1N1 dataset, the training set contains 10,955 true PPIs between human and any virus except H1N1 virus, plus an equal amount (10,955) negative interaction samples. The testing set contains 381 true PPIs between human and H1N1 virus, and 381 negative interactions. Similarly, in the Ebola dataset, the training set contains 11,341 true PPIs between human and any virus except Ebola virus, plus an equal amount (11,341) negative interaction samples. The testing set contains 150 true PPIs between human and Ebola virus, and 150 negative interactions.

Baseline. We compare to the SVM baseline from Zhou et al. [2], which showed that their method was the state-of-the-art.

Results. Table 4.4, shows the results from the Zhou et al. datasets. For both the H1N1 and Ebola datasets, we can see that our method outperforms the previous state-of-the-art. While we believe this dataset is not indicative of a real novel virus setting since the test set negatives are hand-selected, we can use it to compare different methods. Since there is an even positive/negative testing split, AUROC is a good metric to compare

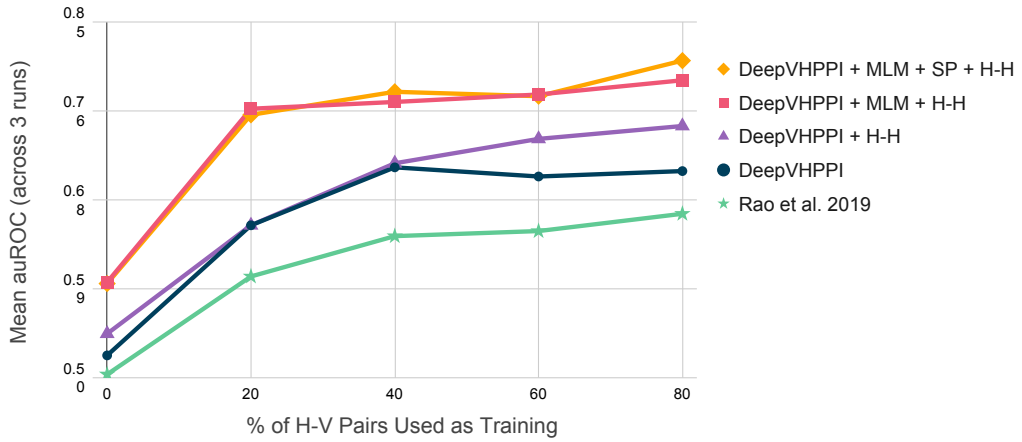

Figure 4.4: Sensitivity analysis on Human–SARS-CoV-2 V-H-PPI Predictions. The X-axis shows simulated training settings, where we assume there exist some (varying) portion of SARS-CoV-2 proteins in the training data. We can see that pretraining methods (LM and SPT) give substantial increases over cases without the pretraining methods. This indicates that transfer learning can help on novel virus protein interaction prediction.

methods, and we can see that across both novel viruses, our method outperforms the SVM. We see notable performance increase using LMT and SP pretraining.

4.5.5 Additional PPI Experiments

We also report the results for our model on two extra baseline datasets. The first is the Virus–Host datasets from Barman et al. [3]. Although the testing dataset is not for a completely unseen test virus, we see our method outperforms the baselines. In Table 4.5, we see that our method outperforms previous methods including an SVM and random forest.

The second is the SLiMs dataset from Eid et al. [4]. This dataset was constructed specifically to evaluate how well the model learns Short Linear Motifs (SLiMs) that are transferable across train/test splits. In Table 4.6, we see that our model is significantly better than the baselines. We hypothesize that this improvement is in part due to the convolutional motif finder layers of DeepVHPPI.

4.5.6 Sensitivity Analysis using Known H-V Interactions

There exist two important situations we need to predict V-H PPI for virus protein with new sequences. (I) The first case is when a new virus is discovered, the protein interactions are unknown. (II) The second case is when the virus is known, and some interactions have been validated, but many are missing. Our above experiments have mostly focused on the first case. Here we experiment and analyze when some of the virus

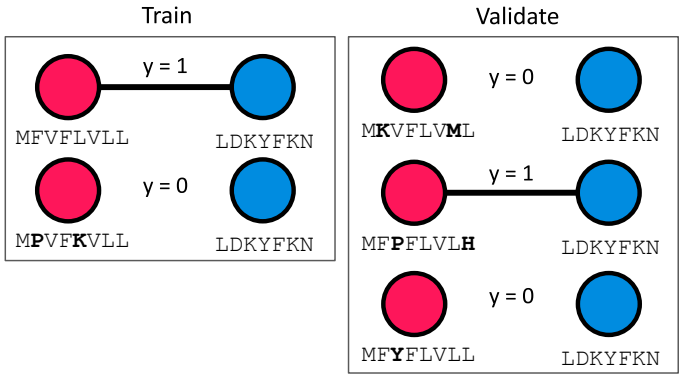

Figure 4.5: Mutation Analysis Setup. We train DeepVHPPI on a subset of known Spike protein mutations and their corresponding ACE2 binding affinity scores, and test on the remaining.

proteins are known. This setup can be used to expand the initial set of experimental interactions, resulting in a more complete interactome.

On the SARS-CoV-2–Human PPI task, we simulate a new setup to evaluate our framework across these two settings. We generate five simulated settings where we vary the percentage of total SARS-CoV-2–Human interactions to be used as training data. We then test on the remaining SARS-CoV-2–Human interactions. The training percentages we use are 0%,20%,40%,60%,80%. We also consider training on all known Human–Human (H–H) interactions from the BioGrid database. Note that we are not using any other virus proteins in these simulations. Fig. 4.4 shows the results across each of the five settings for four DeepVHPPI model variations and a deep learning baseline from Rao et al. [95], which uses MLM pretraining.

First, across all training settings, we can see that DeepVHPPI models outperform the baseline [95]. Across all settings, adding the transfer learning pretraining tasks helps significantly. Most importantly, we can see that in the case I (0%) and case II (20%) settings, MLM and SP pretraining help generalize to the target task, where the non-pretrained models fail to be able to classify well. Additionally, adding H–H interactions helps generalize for predicting novel virus sequence interactions.

4.5.7 Mutation Validation Analysis on SARS-CoV-2 Spike

While predicting novel interactions is an important task, virologists are also interested in understanding how the interactions change based on small changes in the protein sequence. This is particularly important for rapidly changing viruses such as SARS-CoV-2 where small changes in the virus protein sequences can have large effects on infection. To this end, we test the accuracy of a computational model to predict interaction binding affinity changes from changes in the protein sequence.

Number of Training Mutation Pairs	Spearman Correlation
100	0.459
1,000	0.720
10,000	0.937

Table 4.7: Mutation Analysis

We used the deep mutational scan data from Starr et al. [139] to analyze the effectiveness of DeepVHPPI to accurately model binding affinity of virus mutations. This dataset contains 105,526 mutated SARS-CoV-2 Spike receptor binding domain sequences, with the corresponding Human ACE2 dissociation constant. We train the DeepVHPPI + LM model using a subset of the mutation sequences. In other words, the only supervised training is done using a subset of mutation sequence pairs. We consider three different amounts of mutation pairs to use as training: 100, 1,000 and 10,000. The remaining sequences are used as testing. This is demonstrated in Figure 4.5.

Table 4.7 shows the mutation binding accuracy results. We report the Spearman correlation of the continuous valued prediction with the ground truth dissociation constant. As the number of training mutation sequences increases, the Spearman rank correlation with the ground truth dissociation constant increases. By training on only 100 Spike mutation sequences, we can obtain a Spearman correlation of 0.459, and training with 10,000 mutation sequences results in a Spearman correlation of 0.937. This shows that with a relatively few amount of mutation binding affinities to train on, we can get a reasonably good prediction model. Moving forward, we believe this can be used to rapidly analyze mutations and potentially determine how infectious an emerging variant will be on average.

Furthermore, our method allows for an easy way to rapidly test new mutations. Rather than experimentally checking all possible mutations to see which ones reduce interaction binding, we can computationally introduce mutations and observe how the predicted output changes. We show an example of this for a receptor-binding domain subsequence of the SARS-CoV-2 Spike protein when binding to the human ACE2 protein in Fig. 4.6. We can observe specific locations, such as the first "K" amino acid in the virus sequence where mutating the amino acid will reduce the interaction prediction. There are also other locations where mutations can *increase* interaction binding, which may explain how certain viruses are able to mutate and infect humans more easily.

4.5.8 Ablation Study

To investigate the importance of the motif finding convolutional modules of our Transformer architecture, we conducted an ablation study for each model component. Table 4.8 shows the results for the H1N1 and

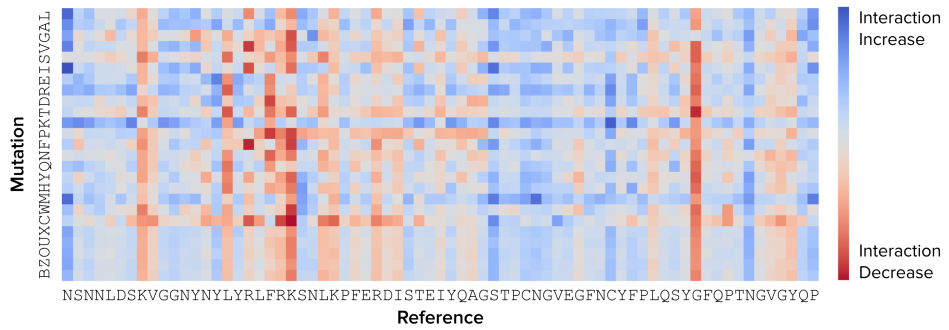

Figure 4.6: **Mutation map for SARS-CoV-2 Spike when interacting with Human ACE2.** Here we show how the predicted interaction score changes when inducing the mutation in the Y-axis for the original amino acid in the X-axis. This example is from the receptor-binding domain in the SARS-CoV-2 Spike protein, and the output shows the predicted difference in interaction score from the original reference amino acid. For example, changing the second reference "K" results in an interaction decrease.

	H1N1		**Ebola**	
Method	AUROC	AUPR	AUROC	AUPR
Transformer [5]	0.922	0.915	0.940	0.949
Convolution Transformer	0.939	0.942	0.955	0.964
Transformer + MLM [5]	0.950	0.957	0.975	0.976
Convolution Transformer + MLM	**0.957**	**0.962**	**0.976**	**0.979**

Table 4.8: Ablation Study. We analyze the effectiveness of the convolution modules in our proposed Transformer architecture compared to a tranditional character level embedding from Rives et al. [5]. Both with and without language model pretraining, the convolution modules result in improved accuracy across the two datasets tested.

Ebola datasets. Across all 4 models tested, we use the 12-layer Transformer architecture from Rives et al. [5]. We ablate the the convolutional motif finding module both with and without language model pretraining. Note that the Transformer + LM model [5] is pretrained using a larger unlabeled dataset (pfam) than our DeepVHPPI Transformer + LM (swissprot). We can see that the convolutional module results in improved AUROC and AUPR with and without language model pretraining. This indicates that protein binding is highly reliant on local, translation invariant features that are easily detected by convolutional filters.

4.6 Discussion and Extensions

While we have focused on predicting the interactions of novel virus sequences and their mutations, we have not considered the variations and profile on the human side of the interactions. Moving forward, there are many opportunities to include the sequence and expression profiles of humans to predict the interaction and infection rate of a virus variant.

We've primarily showed the importance and success of using sequence based models that incorporate protein structure and semantic information for predicting interactions. With the recent success of Alphafold

[140], which is used to predict the full 3D structure of an individual protein from sequence, we believe there are a lot of opportunities to use these models to not only predict the interactions of proteins with other proteins, but also with small molecule drugs and with DNA.

4.7 Summary

Computational methods predicting protein-protein interactions (PPIs) can play an important role in understanding a novel virus that threatens widespread public health. Most previous methods are developed for intra-species interactions, and do not generalize to novel viruses. In this paper, we introduce DeepVHPPI, a novel deep learning architecture that uses a transfer learning approach for PPI prediction between a novel virus and a host. We show that transfer learning and Transformers can be used to accurately predict protein-protein interaction. We demonstrate that our method can help accurately predict Virus–Host interactions early on in the virus discovery and experimentation pipeline. This can help biologists better understand how the virus attacks the human body, allowing researchers to potentially develop effective drugs more quickly. By providing a computational model for interaction prediction, we hope this will accelerate experimental efforts to define a reliable network of Virus–Host protein interactions, as well as predict how threatening a new virus variant may be to public health. While this work is focused on SARS-CoV-2, H1N1, and Ebola, our framework is applicable for any virus. In the case of a future novel virus, our framework will be able to rapidly predict protein-protein interaction predictions.

Chapter 5

Modeling Label Interactions with C-Tran

Understanding visual scenes is a hallmark of human intelligence. Comprehending the contents of an image amounts to understanding the objects and their interactions. Multi-label image classification is the task of predicting a set of labels corresponding to objects, attributes or other entities present in an image. A computational model for multi-label image classification requires a correct set of modeling assumptions to handle both the recognition of objects as well as their dependencies. Furthermore, a flexible model should be able to include and leverage a potential set of "known", or contextually given, labels to more accurately predict the unknown labels.

Research Question 3: *Can we model interactions between object labels, and also utilize the interactions from arbitrary amounts of known labels?*

In this chapter, we propose the Classification Transformer (C-Tran), a general framework for multi-label image classification that leverages Transformers to exploit the complex interactions among visual features and labels. Our approach consists of a Transformer encoder trained to predict a set of target labels given an input set of masked labels, and visual features from a convolutional neural network. Our model shows state-of-the-art performance on challenging datasets such as COCO and Visual Genome. Moreover, our framework is designed for the incorporation of an arbitrary set of known labels to be used as input to the model. This lets the model exploit learned interactions and make stronger predictions on the unknown labels in the image. This also allows for users to interact with the model and test counterfactuals such as "How do the other labels' predictions change if label y_i were present (or absent) in this image?".

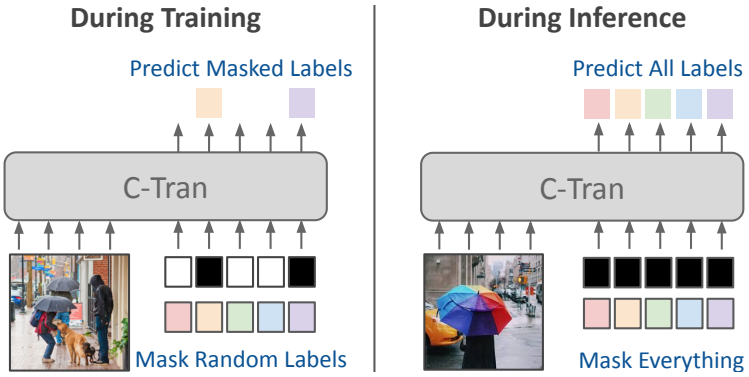

Figure 5.1: **C-Tran training and inference**. We propose a transformer-based model for multi-label image classification that exploits dependencies among a target set of labels using an encoder transformer. During training, the model learns to reconstruct a partial set of labels given randomly masked input label embeddings and image features. During inference, our model can be conditioned only on visual input or a combination of visual input and partial labels, leading to superior results.

5.1 Introduction

Images in real-world applications generally portray many objects and complex situations. Multi-label image classification is a visual recognition task that aims to predict a set of labels corresponding to objects, attributes, or actions given an input image [38, 37, 141, 142, 143, 144, 145]. This task goes beyond the more thoroughly studied problem of single-label multi-class classification where the objective is to extract and associate image features with a single concept per image. In the multi-label setting, the output set of labels has some structure that reflect the structure of the world. For example, *dolphin* is unlikely to co-occur with *grass*, while *knife* is more likely to appear next to a *fork*. Effective models for multi-label classification aim to extract good visual features that are predictive of image labels, but also exploit the complex relations and dependencies between visual features and labels, and among labels themselves.

To this end, we present the Classification Transformer (C-Tran), a multi-label classification framework that leverages a Transformer encoder [19]. Transformers have demonstrated a remarkable capability of being able to exploit dependencies among sets of inputs using multi-headed self-attention layers. In our approach, a Transformer encoder is trained to reconstruct a set of target labels given an input set of masked label embeddings and a set of features obtained from a convolutional neural network. C-Tran uses label masking during training to represent the state of the labels as *positive*, *negative*, or *unknown* – analogous to how language models are trained with masked tokens [50]. At test time, C-Tran is able to predict a set of target labels using only input visual features by masking all the input labels as *unknown*. Figure 5.1 gives an overview of this strategy. We demonstrate that this approach leads to superior results on a number of benchmarks compared to other recent approaches that exploit label relations using graph convolutional

networks and other recently proposed strategies.

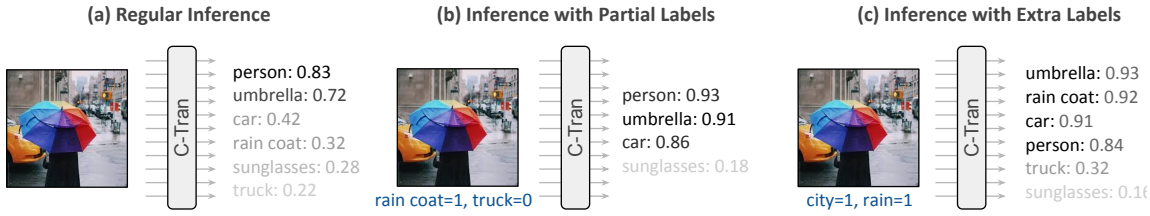

Figure 5.2: **C-Tran inference settings.** Three different inference settings for general multi-label image classification: (a) Standard multi-label classification takes only image features as input. All labels are unknown $\mathbf{y}_{u\cdot}$; (b) Classification under partial labels takes as input image features as well as a subset of the target labels that are known. The labels *rain coat* and *truck* are known labels $\mathbf{y}_k$, and all others are unknown labels $\mathbf{y}_u$; (c) Classification under extra labels takes as input image features and some related extra information. The labels *city* and *rain* are known extra labels $\mathbf{y}^e_k$, and all others are unknown target labels $\mathbf{y}^t_u$.

Beyond obtaining state-of-the-art results on standard multi-label classification, C-Tran is a more general model for reasoning under prior label observations. Because our approach explicitly models the label state (positive, negative, or unknown) during training, it can also be used at test time with partial or extra label annotations by setting the state of some of the labels as either *positive* or *negative* instead of masking them out as *unknown*. For instance, consider the example shown in Figure 5.2(a) where a model is able to predict *person* and *umbrella* with relatively high accuracies, but is not confident for categories such as *rain coat*, or *car* that are clearly present. Suppose we know some labels and set them to their true positive (for *rain coat*) or true negative (for *truck*) values. Provided with this new information, the model is able to predict *car* with a high confidence as it moves mass probability from *truck* to *car*, and predicts other objects such as *umbrella* with even higher confidence than in the original predictions (Figure 5.2(b)).

In general, we consider this setting as realistic since many images also have metadata in the form of extra labels such as location or weather information (Figure 5.2(c)). This type of conditional inference is a much less studied problem. C-Tran is able to naturally handle all these scenarios under a unified framework. We compare our results with a competing method relying on iterative inference [146], and against sensitive baselines, demonstrating superior results under variable amounts of partial or extra labels.

The benefits of C-Tran can be summarized as follows:

- Flexibility: It is the first model that can be deployed in multi-label image classification under arbitrary amounts of extra or partial labels. We use a unified model architecture and training method that lets users to apply our model easily in any setting.

- Accuracy: We evaluate our model on six datasets across three inference settings and achieve state-of-the-art results on all six. The label mask training strategy enhances the correlations between visual

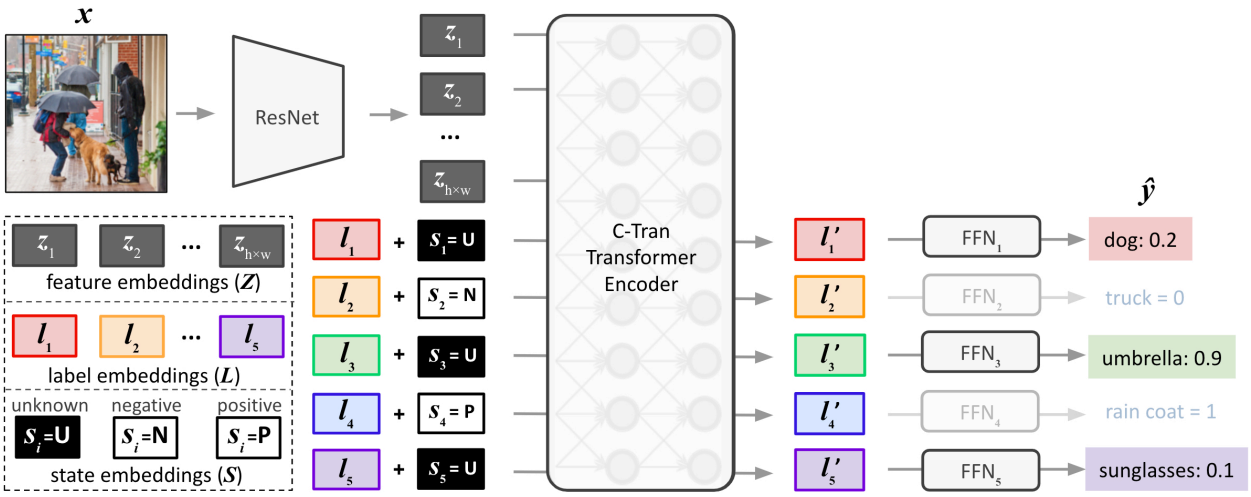

Figure 5.3: **C-Tran architecture**. Model overview and illustration of label mask training for general multi-label image classification. In this training image, the labels *person*, *umbrella*, and *sunglasses* were randomly masked out and used as the unknown labels, $\mathbf{y}_u$. The labels *rain coat* and *truck* are used as the known labels, $\mathbf{y}_k$. Each unknown label is added the unknown state embedding U, and each known label is added its corresponding state embedding: negative (N) , or positive (P). The loss function is only computed on the unknown label predictions $\hat{\mathbf{y}}_u$.

concepts leading to more accurate predictions.

- Interactivity: The use of state embeddings enables users to easily interact with the model and test any counterfactuals. C-Tran can take human interventions as partial evidence and provides more interpretable and accurate predictions.

5.2 Problem Setup

We consider three multi-label image classification scenarios as follows:

Regular Multi-label Classification. In this setting the goal is to predict a set of labels for an input image. Let $\mathbf{x}$ be an image, and $\mathbf{y}$ be a ground truth set of ℓ binary labels $\{y_1, y_2, ..., y_\ell\}, y_i \in \{0, 1\}$. The goal of multi-label classification is to construct a classifier, f, to predict a set of labels given an image so that: $\hat{\mathbf{y}} = f(\mathbf{x})$.

Inference with Partial Labels. While regular classification methods aim to predict the full set of ℓ labels given only an input image, some subset of labels $\mathbf{y}_k \subseteq \mathbf{y}$ may be observed, or known, at test time. This is also known as having partial labels available. For example, many images on the web are labeled with text such as captions or comments on social media. In this reformulated setting, the goal is to predict the unknown labels ($\mathbf{y}_u = \mathbf{y} \setminus \mathbf{y}_k$) given both the image *and* the known labels during inference: $\hat{\mathbf{y}}_u = f(\mathbf{x}, \mathbf{y}_k)$. Note that we assume that all labels are available during training. This setting is specifically for *inference* with partially

annotated labels, and it differs from other works that tackle the problem of training models from partially annotated data [147, 148, 149].

Inference with Extra Labels. Similar to partially labeled images, there are many cases where we observe extra labels that describe an image, but are not part of the target label set. For example, we may know that an image was taken in a *city*. While *city* might not be one of the target labels, it can still alter our expectations about what else might be present in the image. In this setting, we append any extra labels $\mathbf{y}^e$ to the target label set $\mathbf{y}^t$. If there are ℓ^t target labels, and ℓ^e extra labels, we have a set of $\ell^t + \ell^e$ total labels that we use to train the model. Variable $\mathbf{y}$ now represents the concatenation of all target and extra labels. During inference, the known labels, $\mathbf{y}_k^e$, come from the set of extra labels, but we are only interested in evaluating the unknown target labels $\mathbf{y}_u^t$. In other words, during inference, we want to compute the following: $\hat{\mathbf{y}}_u^t = f(\mathbf{x}, \mathbf{y}_k^e)$.

5.3 Method: C-Tran

We propose Classification Transformers (C-Tran), a general multi-label classification framework that works in all three previously described settings. During inference, our method predicts a set of unknown labels $\mathbf{y}_u$ given an input image $\mathbf{x}$ and a set of known labels $\mathbf{y}_k$. In regular inference no labels are known, in partial label inference some labels are known, and in extra label inference some labels external to the target set are known. In Sections 5.3.1-5.3.3, we introduce the C-Tran architecture, and in Section 5.3.4, we explain our label mask training procedure.

5.3.1 Feature, Label, and State Embeddings

Image Feature Embeddings $\boldsymbol{Z}$: Given input image $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$, the feature extractor outputs a tensor $\boldsymbol{Z} \in \mathbb{R}^{h \times w \times d}$, where $h, w,$ and d are the output height, width, and channel, respectively. We can then consider each vector $\boldsymbol{z}_i \in \mathbb{R}^d$ from $\boldsymbol{Z}$, with i ranging from 1 to P (where $P = h \times w$), to be representative of a subregion that maps back to patches in the original image space.

Label Embeddings $\boldsymbol{L}$: For every image, we retrieve a set of label embeddings $\boldsymbol{L} = \{\boldsymbol{l}_1, \boldsymbol{l}_2, ..., \boldsymbol{l}_\ell\}$, $\boldsymbol{l}_i \in \mathbb{R}^d$, which are representative of the ℓ possible labels in $\mathbf{y}$. Label embeddings are learned from an embedding layer of size $d \times \ell$.

Adding Label Knowledge via State Embeddings $\boldsymbol{S}$: In traditional architectures, there is no mechanism to encode partially known or extra labels as input to the model. To address this drawback, we propose

a technique to easily incorporate such information. Given label embedding $\boldsymbol{l}_i$, we simply add a "state" embedding vector, $\boldsymbol{s}_i \in \mathbb{R}^d$:

$$\tilde{\boldsymbol{l}}_i = \boldsymbol{l}_i + \boldsymbol{s}_i, \tag{5.1}$$

where the $\boldsymbol{s}_i$ takes on one of three possible states: unknown (U), negative (N), or positive (P). For instance, if label y_i is a known positive value prior to inference (meaning that we have prior knowledge that the label is present in the image), $\boldsymbol{s}_i$ is the positive embedding, P. The state embeddings are retrieved from a learned embedding layer of size $d \times 3$, where the unknown state vector (U) is fixed with all zeros.

State embeddings enable a user to (1) not use any prior information by adding the unknown embedding, (2), use partially labeled or extra information by adding the negative and positive embeddings to those labels, and (3) easily test interventions in the model by asking "how does the prediction change if a label is changed to either positive or negative?". We note that using prior information is completely optional as input to our model during testing, enabling it to also flexibly handle the regular inference setting.

5.3.2 Modeling Feature and Label Interactions with a Transformer Encoder

To model interactions between image features and label embeddings we leverage Transformers [19], as these are effective models for capturing dependencies between variables. Our formulation allows us to easily input image features and label embeddings jointly into a Transformer encoder. Transformer encoders are suitable because they are order invariant, allowing for any type of dependencies between all features and labels to be learned.

Let $H = \{\boldsymbol{z}_1, ..., \boldsymbol{z}_{h\times w}, \tilde{\boldsymbol{l}}_1, ..., \tilde{\boldsymbol{l}}_\ell\}$ be the set of embeddings that are input to the Transformer encoder. In Transformers, the importance, or weight, of embedding $\boldsymbol{h}_j \in H$ with respect to $\boldsymbol{h}_i \in H$ is learned through *self-attention*. The attention weight, α_{ij}^t between embedding i and j is computed in the following manner. First, we compute a normalized scalar attention coefficient α_{ij} between embeddings i and j. After computing α_{ij} for all i and j pairs, we update each embedding $\boldsymbol{h}_i$ to $\boldsymbol{h}_i'$ using a weighted sum of all embeddings followed by a nonlinear ReLU layer:

$$\alpha_{ij} = \text{softmax}\big((\mathbf{W}^q\boldsymbol{h}_i)^\top(\mathbf{W}^k\boldsymbol{h}_j)/\sqrt{d}\big), \tag{5.2}$$

$$\bar{\boldsymbol{h}}_i = \sum_{j=1}^{M} \alpha_{ij}\mathbf{W}^v\boldsymbol{h}_j, \tag{5.3}$$

$$\boldsymbol{h}_i' = \text{ReLU}(\bar{\boldsymbol{h}}_i\mathbf{W}^r + \boldsymbol{b}_1)\mathbf{W}^o + \boldsymbol{b}_2, \tag{5.4}$$

where $\mathbf{W}^k$ is the key weight matrix, $\mathbf{W}^q$ is the query weight matrix, $\mathbf{W}^v$ is the value weight matrix, $\mathbf{W}^r$ and $\mathbf{W}^o$ are transformation matrices, and $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are bias vectors. This update procedure can be repeated for L layers where the updated embeddings $\boldsymbol{h}'_i$ are fed as input to the successive Transformer encoder layer. The learned weight matrices $\{\mathbf{W}^k, \mathbf{W}^q, \mathbf{W}^v, \mathbf{W}^r, \mathbf{W}^o\} \in \mathbb{R}^{d \times d}$ are not shared between layers. We denote the final output of the Transformer encoder after L layers as $H' = \{\boldsymbol{z}'_1, ..., \boldsymbol{z}'_{h \times w}, \boldsymbol{l}'_1, ..., \boldsymbol{l}'_\ell\}$.

5.3.3 Label Inference Classifier

Lastly, after feature and label dependencies are modeled via the Transformer encoder, a classifier makes the final label predictions. We use an independent feedforward network (FFN_i) for final label embedding $\boldsymbol{l}'_i$. FFN_i contains a single linear layer, where weight $\mathbf{w}_i^c$ for label i is a $1 \times d$ vector, and σ is a simoid function:

$$\hat{y}_i = \text{FFN}_i(\boldsymbol{l}'_i) = \sigma\big((\mathbf{w}_i^c \cdot \boldsymbol{l}'_i) + b_i\big) \tag{5.5}$$

5.3.4 Label Mask Training (LMT)

State embeddings (Eq. 5.1) let us easily incorporate known labels as input to C-Tran. However, we want our model to be flexible enough to handle any amount of known labels during inference. To solve this problem, we introduce a novel training procedure called Label Mask Training (LMT) that forces the model to learn label correlations, and allows C-Tran to generalize to any inference setting.

Inspired by the Cloze task [150] and BERT's masked language model training [50] which works by predicting missing words from their context, we implement a similar procedure. During training, we randomly *mask* a certain amount of labels, and use the ground truth of the other labels (via state embeddings) to predict the masked labels. This differs from masked language model training in that we have a fixed set of inputs (all possible labels) and we randomly mask a subset of them for each sample.

Given that there are ℓ possible labels, the number of "unknown" (i.e. masked) labels for a particular sample, n, is chosen at random between 0.25ℓ and ℓ. Then, n unknown labels, denoted $\mathbf{y}_u$, are sampled randomly from all possible labels $\mathbf{y}$. The unknown state embedding is added to each unknown label. The rest are "known" labels, denoted $\mathbf{y}_k$ and the corresponding ground truth state embedding (positive or negative) is added to each. We call these known labels because the ground truth value is used as input to C-Tran alongside the image. Our model predicts the unknown labels $\mathbf{y}_u$, and binary cross entropy is used to update the model parameters. By masking random amounts of unknown labels (and therefore using random amounts of known labels) during training, the model learns many possible known label combinations, and adapts the model to be used with arbitrary amounts of known information.

We mask out at least 0.25ℓ labels for each training samples for several reasons. First, most masked language model training methods mask out around 15% of the words [50, 127]. Second, we want our model to be able to incorporate anywhere from 0 to 0.75ℓ known labels during inference. We assume that knowing more than 75% of the labels is an unrealistic inference scenario. Our label mask training pipeline thus aims to minimize the following loss:

$$L = \sum_{n=1}^{N_{tr}} \mathbb{E}_{p(\mathbf{y}_k)}\{\text{CE}(\hat{\mathbf{y}}_u^{(n)}, \mathbf{y}_u^{(n)})|\mathbf{y}_k\}, \tag{5.6}$$

where CE represents the cross entropy loss function. $\mathbb{E}_{p(\mathbf{y}_k)}(\cdot|\mathbf{y}_k)$ denotes to calculate the expectation regarding the probability distribution of known labels: $\mathbf{y}_k$. We provide an explanation of the LMT algorithm in the Appendix.

5.3.5 Implementation Details

Image Feature Extractor. For fair comparisons, we use the same image size and pretrained feature extractor as the previous state-of-the-art in each setting. For all datasets except CUB, we use the ResNet-101 [133] pretrained on ImageNet [151] as the feature extractor (for CUB, we use the same as [152]). Since the output dimension of ResNet-101 is 2048, we set our embedding size d as 2048. Following [153, 154], images are resized to 640×640 and randomly cropped to 576×576 with random horizontal flips during training. Testing images are center cropped instead. The output of ResNet-101 is an $18\times18\times d$ tensor, so there are a total of 324 feature embedding vectors, $\boldsymbol{z}_i \in \mathbb{R}^d$.

Transformer Encoder. In order to allow a particular embedding to attend to multiple other embeddings (or multiple groups), C-Tran uses 4 attention heads [19]. We use a L=3 layer Transformer with a residual layer [133] around each embedding update and layer norm [28].

Optimization. Our model, including the pretrained feature extractor, is trained end-to-end. We use Adam [128] for the optimizer with betas=$(0.9, 0.999)$ and weight decay=0. We train the models with a batch size of 16 and a learning rate of 10^{-5}. We use dropout with $p = 0.1$ for regularization.

5.4 Experimental Setup and Results

In the following subsections, we explain the datasets, baselines, and results for the three multi-label classification inference settings.

	All							Top 3					
	mAP	CP	CR	CF1	OP	OR	OF1	CP	CR	CF1	OP	OR	OF1
CNN-RNN [141]	61.2	-	-	-	-	-	-	66.0	55.6	60.4	69.2	66.4	67.8
RNN-Attention [142]	-	-	-	-	-	-	-	79.1	58.7	67.4	84.0	63.0	72.0
Order-Free RNN [143]	-	-	-	-	-	-	-	79.1	58.7	67.4	84.0	63.0	72.0
ML-ZSL [144]	-	-	-	-	-	-	-	74.1	64.5	69.0	-	-	-
SRN [155]	77.1	81.6	65.4	71.2	82.7	69.9	75.8	85.2	58.8	67.4	87.4	62.5	72.9
ResNet101 [133]	77.3	80.2	66.7	72.8	83.9	70.8	76.8	84.1	59.4	69.7	89.1	62.8	73.6
Multi-Evidence [156]	-	80.4	70.2	74.9	85.2	72.5	78.4	84.5	62.2	70.6	89.1	64.3	74.7
ML-GCN [145]	83.0	85.1	72.0	78.0	85.8	75.4	80.3	89.2	64.1	74.6	90.5	66.5	76.7
SSGRL [153]	83.8	**89.9**	68.5	76.8	**91.3**	70.8	79.7	**91.9**	62.5	72.7	**93.8**	64.1	76.2
KGGR [154]	84.3	85.6	72.7	78.6	87.1	75.6	80.9	89.4	64.6	75.0	91.3	66.6	77.0
C-Tran	**85.1**	86.3	**74.3**	**79.9**	87.7	**76.5**	**81.7**	90.1	**65.7**	**76.0**	92.1	**71.4**	**77.6**

Table 5.1: Results of *regular inference* on COCO-80 dataset. The threshold is set to 0.5 to compute precision, recall and F1 scores (%). Our method consistently outperforms previous methods across multiple metrics under the settings of all and top-3 predicted labels. Best results are shown in bold. "-" denotes that the metric was not reported.

	All							Top 3					
	mAP	CP	CR	CF1	OP	OR	OF1	CP	CR	CF1	OP	OR	OF1
ResNet101[133]	30.9	39.1	25.6	31.0	61.4	35.9	45.4	39.2	11.7	18.0	75.1	16.3	26.8
ML-GCN [145]	32.6	42.8	20.2	27.5	66.9	31.5	42.8	39.4	10.6	16.8	77.1	16.4	27.1
SSGRL [153]	36.6	-	-	-	-	-	-	-	-	-	-	-	-
KGGR [154]	37.4	47.4	24.7	32.5	**66.9**	36.5	47.2	48.7	12.1	19.4	78.6	17.1	28.1
C-Tran	**38.4**	**49.8**	**27.2**	**35.2**	**66.9**	**39.2**	**49.5**	**51.1**	**12.5**	**20.1**	**80.2**	**17.5**	**28.7**

Table 5.2: Results of *regular inference* on VG-500 dataset. All metrics and setups are the same as Table 5.1. Our method achieves notable improvement over previous methods.

5.4.1 Regular Inference

Datasets. We use two large-scale regular multi-label classification datasets: COCO-80 and VG-500. COCO [157], is a commonly used large scale dataset for multi-label classification, segmentation, and captioning. It contains $122,218$ images containing common objects in their natural context. The standard multi-label formulation for COCO, which we call COCO-80, includes 80 object class annotations for each image. We use $82,081$ images as training data and evaluate all methods on a test set consisting of $40,137$ images. The Visual Genome dataset [158], contains $108,077$ images with object annotations covering thousands of categories. Since the label distribution is very sparse, we only consider the 500 most frequent objects and use the VG-500

	COCO-80				VG-500				NEWS-500				COCO-1000			
Partial Labels Known (ϵ)	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%
Feedbackprop [146]	80.1	80.6	80.8	80.9	29.6	30.1	30.8	31.6	14.7	21.1	23.7	25.9	29.2	30.1	31.5	33.0
C-Tran	**85.1**	**85.2**	**85.6**	**86.0**	**38.4**	**39.3**	**40.4**	**41.5**	**18.1**	**29.7**	**35.5**	**39.4**	**34.3**	**35.9**	**37.4**	**39.1**

Table 5.3: Results of *inference with partial labels* on four multi-label image classification datasets. Mean average precision score (%) is reported. Across four simulated settings where different amounts of partial labels are available (ϵ), our method significantly outperforms the competing method. With more partial labels available, we achieve larger improvement.

Extra Label Groups Known (ϵ)	0%	36%	54%	71%
Standard [152]	82.7	82.7	82.7	82.7
Multi-task [152]	**83.8**	83.8	83.8	83.8
ConceptBottleneck [152]	80.1	87.0	93.0	97.5
C-Tran	**83.8**	**90.0**	**97.0**	**98.0**

Table 5.4: Results of *inference with extra labels* on CUB-312 dataset. We report the accuracy score (%) for the 200 multi-class target labels. We achieve similar or greater accuracy than the baselines across all amounts of known extra label groups.

subset introduced in [154]. VG-500 consists of $98,249$ training images and $10,000$ test images.

Baselines and Metrics. For COCO-80, we compare to ten well known multi-label classification methods. For VG-500 we compare to four previous methods that used this dataset. Referencing previous works [145, 153, 154], we employ several metrics to evaluate the proposed method and existing methods. Concretely, we report the average per-class precision (CP), recall (CR), F1 (CF1) and the average overall precision (OP), recall (OR), F1 (OF1), under the setting that a predicted label is positive if the output probability is greater than 0.5. We also report the mean average precision (mAP). A detailed explanation of the metrics are shown in the Appendix. For fair comparisons to previous works [156, 155], we also consider the setting where we evaluate the Top-3 predicted labels following. In general, **mAP**, **OF1**, and **CF1** are the most important metrics [145].

Results. C-Tran achieves state-of-the-art performance almost across all metrics on both datasets, as shown in Table 5.1 and Table 5.2. Considering that COCO-80 and VG-500 are two widely studied multi-label datasets, absolute mAP increases of 0.8 and 1.0, respectively, can be considered notable improvements. Importantly, we do not use any predefined feature and label relationship information (e.g. pretrained word embeddings). This signals that our method can effectively learn the relationships.

5.4.2 Inference with Partial Labels

Datasets. We use four datasets to validate our approach in the partial label setting. In all four datasets, we simulate four amounts of partial labels during inference. More specifically, for each testing image, we select ϵ percent of labels as known. ϵ is set to 0% / 25% / 50% / 75% in our experiments. ϵ=0% denotes no known labels, and is equivalent to the regular inference setting.

In addition to COCO-80 and VG-500, we benchmark our method on two more multi-label image classification datasets. Wang et al. [146] derived the top 1000 frequent words from the accompanying captions of COCO images to use as target labels, which we call COCO-1000. There are $82,081$ images for training, and $5,000$ images for validation and testing, respectively. We expect that COCO-1000 provides more and

stronger dependencies compared to COCO-80. We also use the NEWS-500 dataset [146], which was collected from the BBC News. Similar to COCO-1000, the target label set consists of 500 most frequent nouns derived from image captions. There are $151,873$ images for training, $10,304$ for validation and $10,451$ for testing.

Baselines and Metrics. Feedback-prop [146] is an inference method introduced for partial label inference that make use of arbitrary amount of known labels. This method backpropagates the loss on the known labels to update the intermediate image representations during inference. We use the LF method on ResNet-101 Convolutional Layer 13 as in [146]. We compute the mean average precision (mAP) score of predictions on unknown labels.

Results. As shown in Table 5.3, C-Tran outperforms Feedbackprop, in all ϵ percentages of partially known labels on all datasets. In addition, as the percentage of partial labels increases, the improvement of C-Tran over Feedbackprop also increases. These results demonstrate that our method can effectively leverage known labels and is very flexible with the amount of known labels. Feedbackprop updates image features which implicitly encode some notion of label correlation. C-Tran, instead, explicitly models the correlations between labels and features, leading to improved results especially when partial labels are known. On the other hand, Feedback-prop requires careful hyperparameter tuning on a separate validation set and needs time-consuming iterative feature updates. Our method does not require any hyperparameter tuning and just needs a standard one-pass inference. Further comparisons and qualitative examples are included in the Appendix.

5.4.3 Inference with Extra Labels

Datasets. For the extra label setting, we use the Caltech-UCSD Birds-200-2011 (CUB) dataset [159]. It contains 9,430 training samples and 2,358 testing samples. We conduct a multi-classification task with 200 bird species on this dataset. Multi-class classification is a specific instantiation of multi-label classification, where the target classes are mutually exclusive. In other words, each image has only one correct label. We use the processed CUB dataset from Koh et al. [152] where they include 112 extra labels related to bird species. We call this dataset CUB-312. They further cluster extra labels into 28 groups and use varying amounts of known groups at inference time. To make a fair comparison, we consider four different amounts of extra label groups for inference: 0 group (0%), 10 groups (36%), 15 groups (54%), and 20 groups (71%).

Baselines and Metrics. Concept Bottleneck Models [152] incorporate the extra labels as intermediate labels ("concepts" in the original paper). These models use a bottleneck layer to first predict the extra labels, and then use those predictions to predict bird species. I.e., if we let $\mathbf{y}^e$ be the extra information labels, [152] predicts the target class labels $\mathbf{y}^t$ using the following computation graph: $\mathbf{x} \rightarrow \mathbf{y}^e \rightarrow \mathbf{y}^t$. As in [152], we also

consider two baselines: A standard multi-layer perceptron, and a multi-task learning model that predicts the target and concept labels jointly. For fair comparison, we use the same feature extraction method for all experiments, Inception-v3 [160]. We evaluate target predictions using multi-class accuracy scores.

Results. Table 5.4 shows that C-Tran achieves an improved accuracy over Concept Bottleneck models on the CUB-312 task when using any amount of extra label groups. Notably, the multi-task learning model produces the best performing results when ϵ=0. However, it is not able to incorporate known extra labels (i.e., ϵ >0). C-Tran instead, consistently achieves the best performance. Additionally, we can test interventions, or counterfactuals, using C-Tran. For example, "grey beak" is one of the extra labels, and we can set the state embedding of "grey beak" to be positive or negative and observe the change in bird class predictions. We provide samples of extra label interventions in the Appendix.

5.4.4 Ablation and Model Analysis

We conduct ablation studies to analyze the contributions of each C-Tran component. We examine two settings: regular inference (equivalent to 0% known partial labels) and 50% known partial label inference. We evaluate on four datasets: COCO-80, VG-500, NEWS-500, and COCO-1000. First, we remove the image features $\mathbf{Z}$ and predict unknown labels given only known labels. This experiment, C-Tran (no image), tells us how much information model can learn just from labels. Table 5.5 shows that we get relatively high mean average precision scores on some datasets (NEWS-500 and COCO-1000). This indicates that even without image features, C-Tran is able to effectively learn rich dependencies from label annotations .

Second, we remove the label mask training procedure to test the effectiveness of this technique. More specifically, we remove all label state embeddings, $\mathbf{S}$; thus all labels are unknown during training. Table 5.5 shows that for both settings, regular (0%) and 50% partial labels known, the performance drops without label mask training. This signifies two critical findings of label mask training: (1) it helps with dependency learning as we see improvement when no partial labels are available during inference. This is particularly true for datasets that have strong label co-occurrences, such as NEWS-500 and COCO-1000. (2) given partial labels, it can significantly improve prediction accuracy. We provide a t-SNE plot [161] of the label embeddings learned with and without label mask training. As shown in Figure 5.4, embeddings learned with label mask training exhibit a more meaningful semantic topology; i.e. objects belonging to the same group are clustered together.

We also analyze the importance of the number of Transformer layers, L, for regular inference in COCO-80. Mean average precision scores for 2, 3, and 4 layers were 85.0, 85.1, and 84.3, respectively. This indicates : (1)

Partial Labels	COCO-80		VG-500		NEWS-500		COCO-1000	
Known (ϵ)	0%	50%	0%	50%	0%	50%	0%	50%
C-Tran (no image)	3.60	21.7	2.70	24.6	6.50	33.3	1.50	27.8
C-Tran (no LMT)	84.8	85.0	38.3	38.8	16.9	17.1	33.1	34.0
C-Tran	**85.1**	**85.6**	**38.4**	**40.4**	**18.1**	**35.5**	**34.3**	**37.4**

Table 5.5: C-Tran component ablation results. Mean average precision score (%) is reported. Our proposed Label Mask Training technique (LMT) improves the performance, especially when partial labels are available.

our method is fairly robust to the number of Transformer layers, (2) multi-label classification does not seem to require a very large number of layers as in some NLP tasks [127]. While we show C-Tran is a powerful method in many multi-label classification settings, we recognize that Transformer layers are memory-intensive for a large number of inputs. This limits the number of possible labels ℓ in our model. Using four NVIDIA Titan X GPUs, the upper bound of ℓ is around 2000 labels. However, it is possible to increase the number of labels. We currently use the ResNet-101 output channel size ($d = 2048$) for our Transformer hidden layer size. This can be linearly mapped to a smaller number. Additionally, we could apply one of the Transformer variations that have been proposed to model very large input sizes [51, 162].

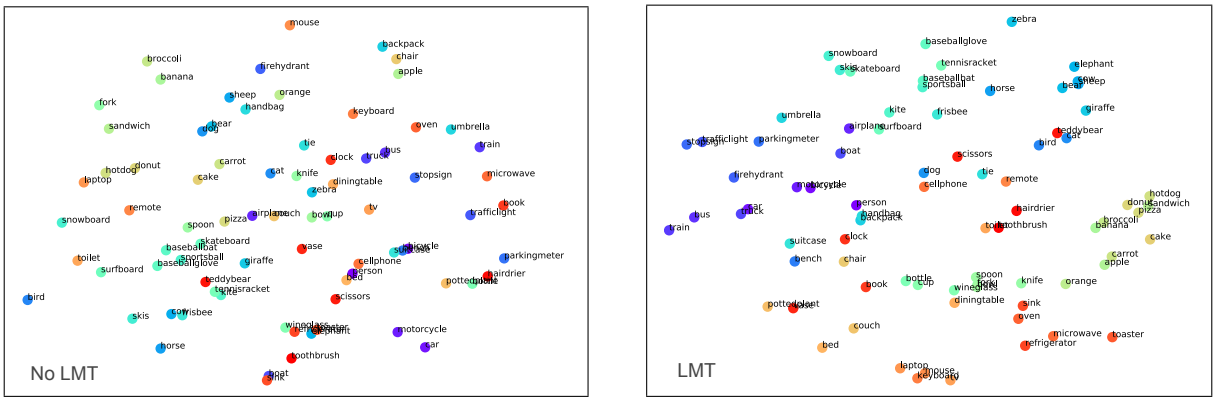

Figure 5.4: Comparison of the learned label embeddings for COCO-80 using t-SNE. The left figure shows the embedding projections without using label mask training (LMT), and the right shows with LMT. Labels are colored using the COCO object categorization. We can see that using label mask training produces much semantically stronger label representations.

5.4.5 Qualitative Examples

Inference with Partial Labels. In Figure 5.8, we show qualitative results on COCO-80 demonstrating the use of partial labels. In these examples, we first show the predictions for ResNet-101, as well as C-Tran without using partial labels. The last column shows the C-Tran predictions when using $\epsilon = 25\%$ partial labels (which is 21 labels for COCO-80) as observed, or known prior to inference. For many examples, certain labels cannot be predicted well without using partial labels.

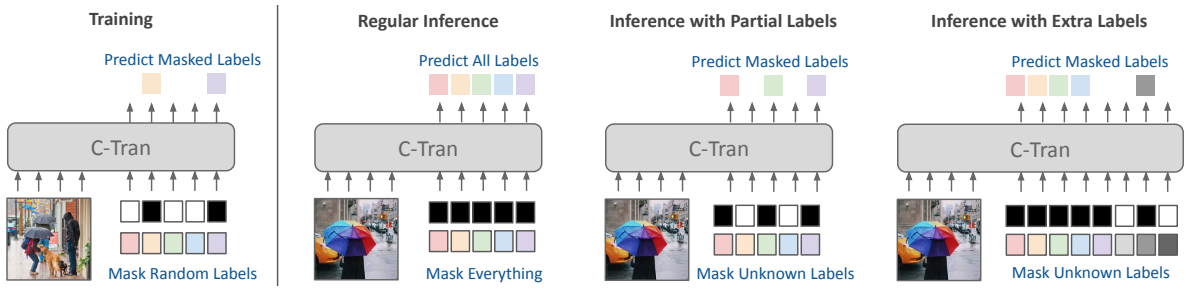

Figure 5.5: **Detailed training and inference settings.** Detailed illustrations of the general training method and three different inference settings where C-Tran can be applied.

Inference with Extra Labels. In Figure 5.9, we show qualitative results on CUB-312 demonstrating the use of extra labels. In the CUB-312 dataset, the extra labels are high level concepts of bird species that are not target labels. In these examples, we first show the predictions for C-Tran without using extra labels labels, and the last column shows the C-Tran predictions when using $\epsilon = 54\%$ of the extra labels (which is 60 labels for CUB-312) as observed, or known prior to inference. We can see that many bird species predictions are completely changed after using the extra labels as input to our model.

5.4.6 Detailed Diagram of C-Tran Settings

Figure 5.5 shows a detailed diagram of all possible training and inference settings used in our paper, and how C-Tran is used in each setting. By using the same random mask training, we can apply our model to any of the three inference settings.

5.4.7 Label Mask Training

Algorithm 1 C-Tran Label Mask Training Procedure

loss = 0
for `sample (x, y) in batch` **do**
    label_idxs = range(1, ℓ);
    n = randint(0.25ℓ,ℓ);
    unk_idxs = sample(label_idxs, n);
    $\mathbf{y}_u = \{y_i$ for i in unk_idxs$\}$;
    $\mathbf{y}_k = \{y_j$ for j in in label_idxs excluding unk_idxs$\}$;
    $\hat{\mathbf{y}}_u = f(\mathbf{x}, \mathbf{y}_k; \theta)$;
    for `label index j in` $\mathbf{y}_u$ **do**
        loss += $-(y_j \log(\hat{y}_j) + (1 - y_j) \log(1 - \hat{y}_j))$
    end for
end for
backprop;
Update θ;

Attribute Value	Class Prediction
has_yellow_underparts=1	Heermann_Gull
has_yellow_underparts=0	Glaucous_windged_Gull

Figure 5.6: **Counterfactual example.** The ground truth is *Heermann Gull*. If we incorporate the "has yellow underparts" attribute as input to the model, it correctly predicts the Glaucous-winged Gull bird class.

In Algorithm 1, we detail the label mask training (LMT) procedure. For each training sample, we select a random amount of labels to be used as "known" input labels to the model. The loss function is then computed on all unknown labels.

5.4.8 Counterfactual Testing

Counterfactual testing, as introduced by Koh et al., is performed to answer the question "If I know that some label is true, how does it change the prediction of other labels?". Fig. 5.6 shows a counterfactual example on the CUB-312 dataset. Here we show the bird class prediction of our model contingent on *has_ yellow_ underparts* being true (=1) or false (=0). In other words, this allows the user to answer the question "What kind of bird would this be if it has (or doesn't have) yellow underparts?".

5.4.9 Attention Weight Analysis

In Fig. 5.7 we demonstrate attention weight analysis from the third layer of our model on the COCO-80 dataset. Fig. 5.7 (a), shows label-to-image attention for the "frisbee" label. The frisbee label attends to the frisbee object in the image. Fig. 5.7 (b) shows label-to-label attention. Most labels attend to the frisbee label. We found that sometimes the predictions can be worse if the model relies too much on partial labels, but overall the performance is improved.

5.5 Related Work

Our work relates to the prior literature in image categorization. While a lot of work focuses on single-label classification [151], there is also an ample body of work on both multi-label prediction and exploiting label

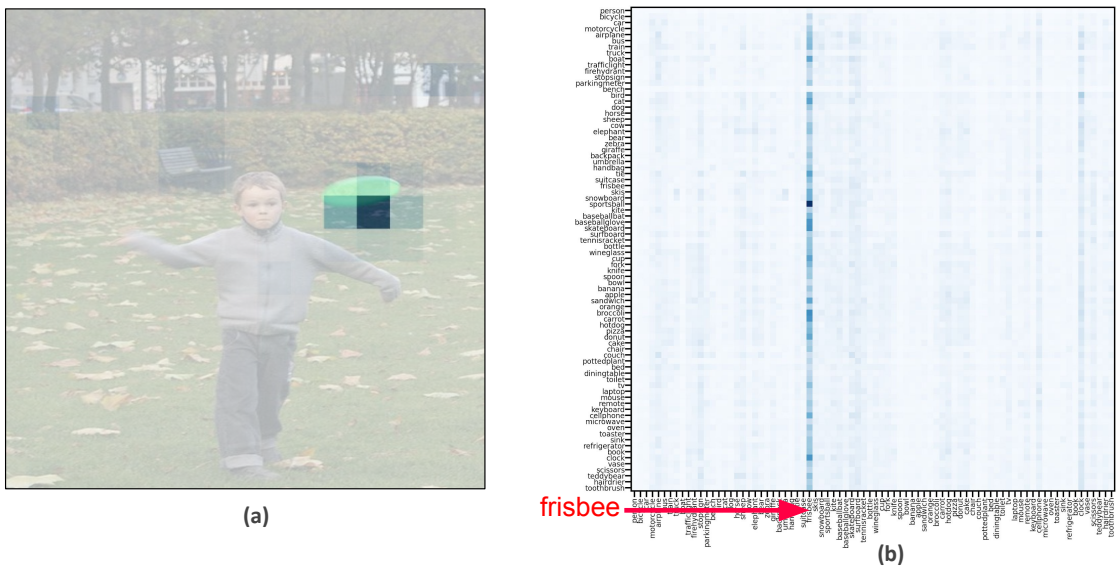

Figure 5.7: **Attention visualization. (a)** Frisbee-to-image attention. The frisbee label embedding attends to the frisbee in the image. **(b)** Label-to-label attention. Most labels attend to the frisbee label.

dependencies [163, 164, 165, 166]. There is also an increasing recognition of the importance of being able to handle partial labels both during training and inference. We review some of this work in this section as follows:

Multi-label Image Classification. Multi-label classification (MLC) is gaining popularity due to its relevance in real world applications. Recently, Stock et al [167] showed that the remaining error in ImageNet is not due to the feature extraction, but rather that ImageNet is annotated with single labels even when some images depict more than one object.

Recent literature addressing multi-label classification roughly fall into four groups. *(1) Conditional Prediction:* The first type, autoregressive models [168, 169, 141, 170] estimate the true joint probability of output labels given the input by using the chain rule, predicting one label at a time. *(2) Shared Embedding Space:* The second group learns to project input features and output labels into a shared latent embedding space [171, 172]. *(3) Structured Output:* The third kind describes label dependencies using structured output inference formulation [173, 174, 175, 176, 177, 178, 179, 180]. *(4) Label Graph Formulation:* Several recent studies [145, 181, 153, 154] used graph neural networks to model label dependency an obtained state-of-the-art results. All methods relied on knowledge-based graphs being built from label co-occurrence statistics. Our proposed model is most similar to *(4)*, but it does not need extra knowledge to build a graph and can automatically learn label dependencies.

Inference with Partial Labels, Wang et al. proposed feedback propagation to handle any set of partial

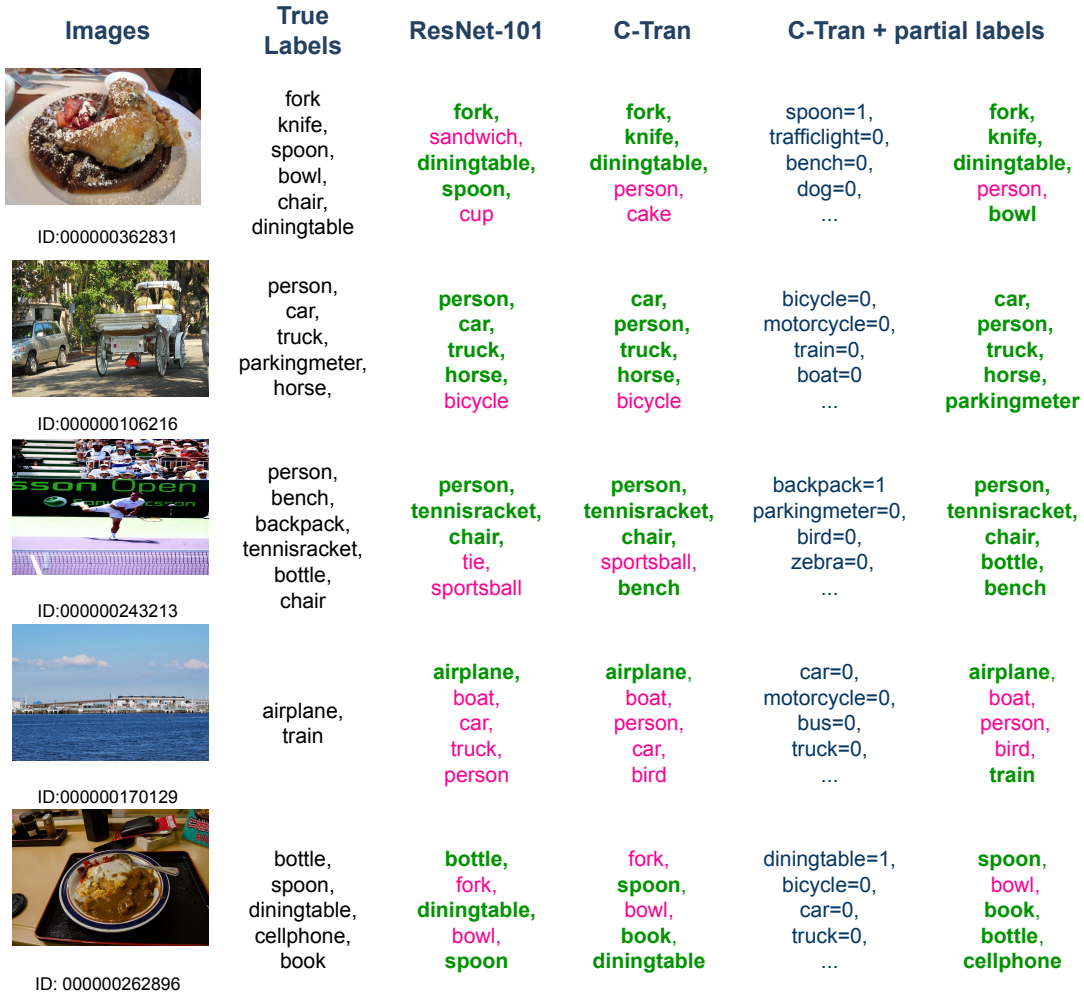

Figure 5.8: **Qualitative examples of C-Tran + partial labels on the COCO-80 dataset.** In the last column, we use $\epsilon = 25\%$ partial labels, some of which are shown. Correctly predicted labels are in bold.

labels at test time [146]. The idea is to optimize intermediate image representations according to *known* labels and then predict *unknown* labels based on updated representations. Yang et al [182] use this type of approach to pivot information across captions in different languages. Huang et al [183] use feedback consistency to improve adversarial robustness. However, these methods require many iterations at inference time, and particularly the model in [146] is not exposed to partial evidence during training, which limits potential gains. Several methods [184, 166] utilize partial labels using a fixed set of labels. In realistic settings, however there could be an arbitrary set of known labels available during inference. If there are ℓ total labels, then the number of known labels, $n{=}|\mathbf{y}_k|$ ranges from 0 to ℓ-1. The number of possible known label sets is then $\binom{\ell}{n}$. C-Tran, integrates a novel representation indicating each label state as *positive*, *negative* or *unknown*. This representation enables us to leverage partial signals during training, and make our model compatible with

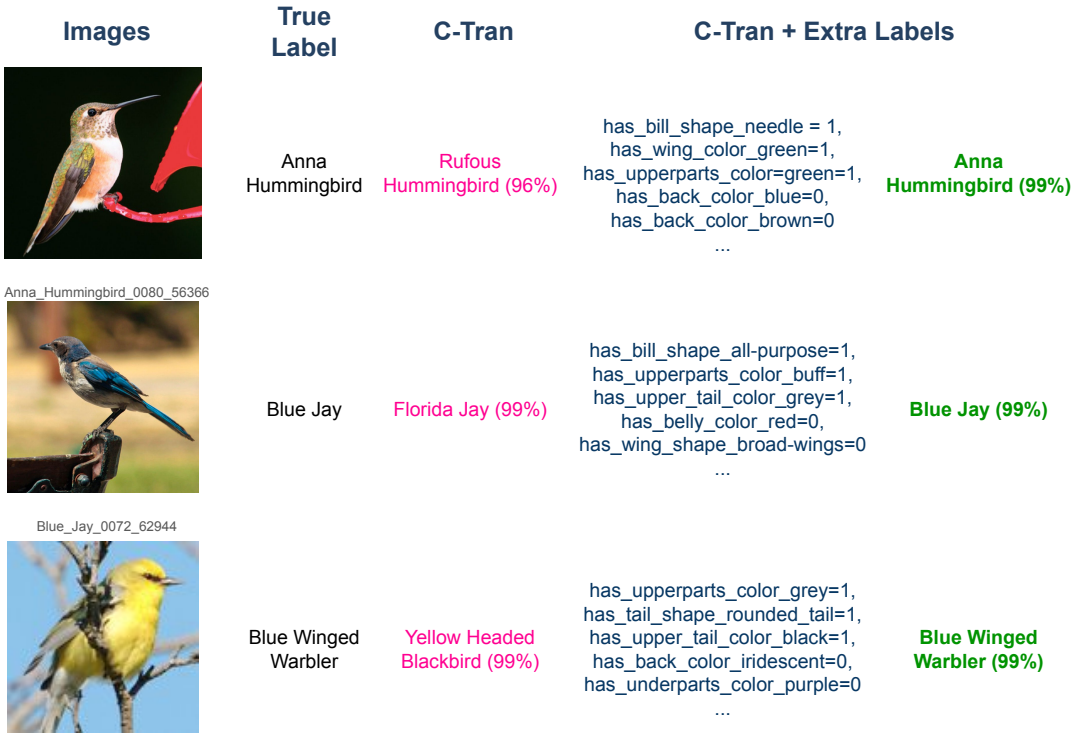

Figure 5.9: **Qualitative examples of C-Tran + extra labels on the CUB-312 dataset.** In the last column, we use $\epsilon = 54\%$ extra labels, some of which are shown.

any known label set during inference. Notably, C-Tran can exploit arbitrary amounts of partial evidence during both training and inference.

Many prominent works also tackle the problem of *training* models with partial label annotations [147, 148, 149]. While this might seem similar to our setting, the key distinction is that these methods assume that images have incomplete or partial labels only during training. However, partial label training methods make no assumptions about the inference settings and thus cannot be easily extended to the scenario where partial labels are available at test time. We consider our line of work complementary to these efforts as these are not mutually exclusive.

Inference with Extra Labels, Koh et al [152] introduces Concept Bottleneck Models which incorporate intermediate concept labels as a bottleneck layer for the target label classification. Similar to [184], this model assumes that the concept labels are a fixed set. Our model goes further by relaxing the need for a fixed set and uses state embeddings instead of a concept bottleneck layer to represent each concept as *known* (positive or negative) or *unknown*. This representation enables C-Tran to leverage partial labels (concepts) during training, and make our model compatible with any known labels (concepts) during inference.

Transformers for Computer Vision Several recent works have used Transformers in computer vision

applications [185, 186, 187, 188, 189]. Some of these models replace a significant part of the visual recognition pipeline with a transformer [188, 187, 185] while others use a transformer on top of features computed by a convolutional neural network [186, 189]. Our model is architecturally similar to the latter, with a focus on using arbitrary amounts of output labels as *input* to the model.

Connecting to Transformers and BERT. Our proposed method, C-Tran, draws much inspiration from works in natural language processing. The transformer model [19] proposed "self attention" for natural language translation. Self attention allows each word in the target sentence to attend to all other words (both in the source sentence and the target sentence) for translation. [50] introduced BERT for language modeling. BERT uses self attention with masked words to pretrain a language model.

Self attention and BERT are both examples of complete graphs, but on sentences rather than image features and labels. C-Tran uses the same self-attention mechanisms as [19] and [50], but instead of using only the word embeddings from a sentence, we use feature and label embeddings.

In computer vision, [185] used Transformers for object detection. Our method varies in several distinct ways. First, we are primarily interested in using partial evidence for image classification, and our unique state embeddings allow C-Tran to use such evidence. Second, we model image and label features jointly in a Transformer encoder, whereas [185] use an encoder/decoder framework. Our method allows the image features to be updated conditioned on the labels, which is a key characteristic of our model.

Connecting to Graph Based Neural Relational Learning. Another line of recent works employ object localization techniques [190, 191] or attention mechanism [192, 155] to locate semantic meaningful regions and try to identify underlying relations between regions and outputs. However, these methods either require expensive bounding box annotations or merely get regions of interest roughly due to the lack of label supervision. One recent study by [193] also showed that modeling the associations between image feature regions and labels helps to improve multi-label performance. In our work, C-Tran uses graph attentions and enables each target label to attend differentially to relevant parts of an input image.

For multi-label classfication(MLC), [176] formulate MLC using a label graph and they introduced a conditional dependency SVM where they first trained separate classifiers for each label given the input and all other true labels and used Gibbs sampling to find the optimal label set. The main drawback is that this method requires separate classifiers for each label. [194] proposes a method to label the pairwise edges of randomly generated label graphs, and requires some chosen aggregation method over all random graphs. The authors introduce the idea that variation in the graph structure shifts the inductive bias of the base learners. One recent study [181] used graph neural networks for multi-label classification on sequential inputs. The

proposed method models the label-to-label dependencies using GNNs, however, does not represent input features and labels in one coherent graph. A key aspect of C-Tran is that the Transformer encoder can be viewed as a fully connected graph which is able to learn any relationships between features and labels. The Transformer attention mechanism can be regarded as a form of graph ensemble learning [195]. Above all, previous methods using graphs to model label dependencies do not allow for partial evidence information to be included in the prediction.

5.6 Discussion and Extensions

We believe that one of the reasons our C-Tran method works well, especially in the partial label setting, is because of the strong interaction dependencies within images. We argue that designing models with these types of interactions in mind is a critical choice moving forward. Furthermore, incorporating contextual information is one of the key steps for designing better classification models. Humans are able to exploit contextual information to make more informed decisions in certain scenarios. We demonstrated the effectiveness of incorporating partial label contextual information, but there are many different modalities for including contextual information. This includes, but is not limited to, proximal images or videos, as well as extra text information. Further work could also extend C-Tran for hierarchical scene categorization, and explore training strategies to make C-Tran generalize to settings where some labels have never been observed during training.

One drawback of our approach is that models which incorporate the dependencies between labels can over-rely on them resulting in biased models [196]. While these approaches can improve accuracy, they can also be detrimental. Further studies to investigate where relying on label interactions is harmful is an important task in deploying these models in real world settings. Similarly, these dependencies can potentially be used to create adversarial examples [197, 198, 199], where we incorrectly tell the model there is a label present (or absent) in order to get it make a wrong prediction. Further studies to investigate and mitigate adversarial attacks in for multi-label classification models are an important research direction moving forward.

5.7 Summary

We propose C-Tran, a novel and flexible deep learning model for multi-label image classification. We show that Transformers can be used to exploit interactions within image scenes for accurate multi-label image classification. Our approach is easy to implement and can effectively leverage an arbitrary set of partial or

extra labels during inference. C-Tran learns sample-adaptive interactions through attention and discovers how labels attend to different parts of an image. We demonstrate the effectiveness of our approach in regular multi-label classification and multi-label classification with partially observed or extra labels. C-Tran outperforms state-of-the-art methods in a wide range of scenarios. We further provide a quantitative and qualitative analysis showing that C-Tran obtains gains by explicitly modeling interactions between target labels and between image features and target labels.

Chapter 6

Conclusion and Future Work

6.1 Intellectual Merit and Broader Impacts

ChromeGCN is a novel application of deep neural networks for predicting and understanding properties of genomic sequences. To our knowledge, ChromeGCN is the first work to extend deep learning models to incorporate long range interactions from Hi-C data. Furthermore, the Deep Motif Dashboard is the first unified visualization framework for extracting interpretable insights from arbitrary deep learning models in genomics. DeepVHPPI is a novel deep learning method for predicting and analyzing novel virus protein interactions with human proteins. To our knowledge, this is the first work to use transfer learning to incorporate protein structural and semantic information for protein-protein interaction prediction. C-Tran is a novel architecture and training method for general multi-label image classification. To our knowledge, we introduce the first deep learning model to incorporate arbitrary amounts of partial labels during inference.

The research in this thesis is a successful fusion of important aspects of biology, computer vision, and machine learning. The primary aim of this work is to redefine and solve challenges within biology and computer vision. At the same time, we further improve the state-of-the-art machine learning methods. With the help of interpretable and fast tools like ChromeGCN, Deep Motif Dashboard, and DeepVHPPI, we hope to provide a better understanding of underlying mechanisms in biology. With C-Tran we hope to provide a unified model for multi-label classification problems where users can better interact with and understand the model and task.

6.2 Paths Forward

ChromeGCN is a general graph neural network framework for incorporating and learning important genomic sequence interactions and we plan to extend it to incorporate gene expression prediction. By training better representation learning models directly from sequence data, we hope to build more realistic simulations of human biology so that biologists can more rapidly and accurately test DNA sequence hypotheses in-silico. These models can be used to both better understand existing genomic pathways, as well as to predict the effects of mutations and gene editing tools such as CRISPR-Cas9 [200].

DeepVHPPI is a general transfer learning framework for predicting protein-protein interactions. We plan to train DeepVHPPI on all possible protein-protein interactions so that the model is applicable for not only virus-human interactions. This model can predict novel protein interactions, predict the effects of mutations and variants, and help understand why protein interactions occur.

C-Tran is a general Transformer framework for multi-label classification. We plan to extend it in multiple directions. First, there are easily transferable applications other domains aside from images, such as text classification. Second, there are other forms of contextual information that we can incorporate in addition to partial labels including surrounding text and prior memories [201]. Finally, an exploration of the trade-offs between the accuracy improvement from using highly dependent interactions and the detrimental effects of incorrect interactions is an important direction.

6.3 Reflections

In this thesis, we've shown that we can use graph neural network models to use and discover important interactions in DNA, transfer learning and Transformers to predict and understand interactions in proteins, and Transformers with context specific information to exploit and explain interactions within image scenes. Our work presents solid evidence about the importance of modeling interactions with deep learning and proposes practical solutions to improve on previous work. The results also help understand the underlying relationships of the data in the tasks explored. In an age where end-to-end learning with the fewest assumptions possible seems to be the goal, we've shown that incorporating structural interaction assumptions into the model can be extremely useful.

Bibliography

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[2] Xiang Zhou, Byungkyu Park, Daesik Choi, and Kyungsook Han. A generalized approach to predicting protein-protein interactions between virus and host. *BMC genomics*, 19(6):568, 2018.

[3] Ranjan Kumar Barman, Sudipto Saha, and Santasabuj Das. Prediction of interactions between viral and host proteins using supervised machine learning methods. *PloS one*, 9(11):e112034, 2014.

[4] Fatma-Elzahraa Eid, Mahmoud ElHefnawi, and Lenwood S Heath. Denovo: virus-host sequence-based protein–protein interaction prediction. *Bioinformatics*, 32(8):1144–1150, 2016.

[5] Alexander Rives, Siddharth Goyal, Joshua Meier, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, page 622803, 2019.

[6] Suhas SP Rao, Miriam H Huntley, Neva C Durand, Elena K Stamenova, Ivan D Bochkov, James T Robinson, Adrian L Sanborn, Ido Machol, Arina D Omer, Eric S Lander, et al. A 3d map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*, 159(7):1665–1680, 2014.

[7] Jian Zhou, Chandra L Theesfeld, Kevin Yao, Kathleen M Chen, Aaron K Wong, and Olga G Troyanskaya. Deep learning sequence-based ab initio prediction of variant effects on expression and disease risk. *Nature genetics*, 50(8):1171, 2018.

[8] Thomas N Kipf. Deep learning with graph-structured representations. 2020.

[9] Joshua B Tenenbaum, Thomas L Griffiths, and Charles Kemp. Theory-based bayesian models of inductive learning and reasoning. *Trends in cognitive sciences*, 10(7):309–318, 2006.

[10] Thomas L Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in cognitive sciences*, 14(8):357–364, 2010.

[11] Tomer D Ullman, Elizabeth Spelke, Peter Battaglia, and Joshua B Tenenbaum. Mind games: Game engines as an architecture for intuitive physics. *Trends in cognitive sciences*, 21(9):649–665, 2017.

[12] Noam Chomsky. *Aspects of the Theory of Syntax*, volume 11. MIT press, 2014.

[13] Wilhelm Humboldt. On language: On the diversity of human language construction and its influence on the mental development of the human species. 1999.

[14] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[15] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[16] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[18] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[20] Tom M Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research ..., 1980.

[21] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[23] Jeffrey L Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine learning*, 7(2-3):195–225, 1991.

[24] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[25] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[27] The transformer model in equations. `https://homes.cs.washington.edu/~thickstn/docs/transformers.pdf`. Accessed: 20121-01-12.

[28] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[29] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[30] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[31] Yanjun Qi. *Learning of protein interaction networks*. Carnegie Mellon University, 2008.

[32] Vicente Ordónez Román. *Language and perceptual categorization in computational visual recognition*. PhD thesis, The University of North Carolina at Chapel Hill, 2015.

[33] David Raposo, Adam Santoro, David Barrett, Razvan Pascanu, Timothy Lillicrap, and Peter Battaglia. Discovering objects and their relations from entangled scene representations. *arXiv preprint arXiv:1702.05068*, 2017.

[34] Adam Santoro, David Raposo, David GT Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *arXiv preprint arXiv:1706.01427*, 2017.

[35] Andrew Kachites McCallum. Multi-label text classification with a mixture model trained by em. In *AAAI 99 workshop on text learning*. Citeseer, 1999.

[36] Naonori Ueda and Kazumi Saito. Parametric mixture models for multi-labeled text. In *Advances in neural information processing systems*, pages 737–744, 2003.

[37] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3), 2006.

[38] André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Advances in neural information processing systems*, pages 681–687, 2002.

[39] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *European conference on machine learning*, pages 406–417. Springer, 2007.

[40] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333, 2011.

[41] Min-Ling Zhang and Zhi-Hua Zhou. A k-nearest neighbor based algorithm for multi-label classification. In *Granular Computing, 2005 IEEE International Conference on*, volume 2, pages 718–721. IEEE, 2005.

[42] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 22–30. Springer, 2004.

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

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

[45] Borbala Mifsud, Filipe Tavares-Cadete, Alice N Young, Robert Sugar, Stefan Schoenfelder, Lauren Ferreira, Steven W Wingett, Simon Andrews, William Grey, Philip A Ewels, et al. Mapping long-range promoter contacts in human cells with high-resolution capture hi-c. *Nature genetics*, 47(6):598, 2015.

[46] Xiaoyan Ma, Daphne Ezer, Boris Adryan, and Tim J Stevens. Canonical and single-cell hi-c reveal distinct chromatin interaction sub-networks of mammalian transcription factors. *Genome biology*, 19(1):174, 2018.

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

[48] Chris A Brackley, Mike E Cates, and Davide Marenduzzo. Facilitated diffusion on mobile dna: configurational traps and sequence heterogeneity. *Physical review letters*, 109(16):168103, 2012.

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

[50] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[51] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[52] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

[53] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.

[54] Ferhat Ay, Timothy L Bailey, and William Stafford Noble. Statistical confidence estimation for hi-c data reveals regulatory chromatin contacts. *Genome research*, 24(6):999–1011, 2014.

[55] Mahmoud Ghandi, Dongwon Lee, Morteza Mohammad-Noori, and Michael A Beer. Enhanced regulatory sequence prediction using gapped k-mer features. *PLoS computational biology*, 10(7):e1003711, 2014.

[56] Ritambhara Singh, Arshdeep Sekhon, Kamran Kowsari, Jack Lanchantin, Beilun Wang, and Yanjun Qi. Gakco: a fast gapped k-mer string kernel using counting. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 356–373. Springer, 2017.

[57] Hamid Reza Hassanzadeh and May D Wang. Deeperbind: Enhancing prediction of sequence specificities of dna binding proteins. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 178–183. IEEE, 2016.

[58] Jack Lanchantin, Ritambhara Singh, Zeming Lin, and Yanjun Qi. Deep motif: Visualizing genomic sequence classifications. *arXiv preprint arXiv:1605.01133*, 2016.

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

[60] Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi. Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks. In *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2017*, pages 254–265. World Scientific, 2017.

[61] David R Kelley, Yakir A Reshef, Maxwell Bileschi, David Belanger, Cory Y McLean, and Jasper Snoek. Sequential regulatory activity prediction across chromosomes with convolutional neural networks. *Genome research*, 28(5):739–750, 2018.

[62] Ofir Hakim and Tom Misteli. Snapshot: chromosome conformation capture. *Cell*, 148(5):1068–e1, 2012.

[63] Swneke D Bailey, Xiaoyang Zhang, Kinjal Desai, Malika Aid, Olivia Corradin, Richard Cowper-Sal, Batool Akhtar-Zaidi, Peter C Scacheri, Benjamin Haibe-Kains, Mathieu Lupien, et al. Znf143 provides sequence specificity to secure chromatin interactions at gene promoters. *Nature communications*, 6:6186, 2015.

[64] Ka-Chun Wong. Motifhyades: expectation maximization for de novo dna motif pair discovery on paired sequences. *Bioinformatics*, 33(19):3028–3035, 2017.

[65] Jacob Schreiber, Maxwell Libbrecht, Jeffrey Bilmes, and William Noble. Nucleotide sequence and dnasei sensitivity are predictive of 3d chromatin architecture. *bioRxiv*, page 103614, 2018.

[66] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pages 2702–2711, 2016.

[67] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

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

[69] Anshul Kundaje, Wouter Meuleman, Jason Ernst, Misha Bilenky, Angela Yen, Alireza Heravi-Moussavi, Pouya Kheradpour, Zhizhuo Zhang, Jianrong Wang, Michael J Ziller, et al. Integrative analysis of 111 reference human epigenomes. *Nature*, 518(7539):317, 2015.

[70] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3145–3153. JMLR. org, 2017.

[71] Timothy L Bailey, Mikael Boden, Fabian A Buske, Martin Frith, Charles E Grant, Luca Clementi, Jingyuan Ren, Wilfred W Li, and William S Noble. Meme suite: tools for motif discovery and searching. *Nucleic acids research*, 37(suppl_2):W202–W208, 2009.

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

[73] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *The Journal of Machine Learning Research*, 11:1803–1831, 2010.

[74] Anthony Mathelier, Oriol Fornes, David J Arenillas, Chih-yu Chen, Grégoire Denay, Jessica Lee, Wenqiang Shi, Casper Shyr, Ge Tan, Rebecca Worsley-Hunt, et al. Jaspar 2016: a major expansion and update of the open-access database of transcription factor binding profiles. *Nucleic acids research*, 44(D1):D110–D115, 2016.

[75] Shobhit Gupta, John A Stamatoyannopoulos, Timothy L Bailey, and William Stafford Noble. Quantifying similarity between motifs. *Genome biology*, 8(2):1–9, 2007.

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

[77] Ritambhara Singh, Jack Lanchantin, Gabriel Robins, and Yanjun Qi. Deepchrome: deep-learning for predicting gene expression from histone modifications. volume 32, pages i639–i648, 2016.

[78] Ritambhara Singh, Jack Lanchantin, Arshdeep Sekhon, and Yanjun Qi. Attend and predict: Understanding gene regulation by selective attention on chromatin. In *Advances in neural information processing systems*, pages 6785–6795, 2017.

[79] Sylvain Pitre, Mohsen Hooshyar, Andrew Schoenrock, Bahram Samanfar, Matthew Jessulat, James R Green, Frank Dehne, and Ashkan Golshani. Short co-occurring polypeptide regions can predict global protein interaction maps. *Scientific reports*, 2:239, 2012.

[80] EM Phizicky and S. Fields. Protein-protein interactions: methods for detection and analysis. *Microbiol Rev.*, 59(1):94–123, 1995.

[81] Shao-Wu Zhang and Ze-Gang Wei. Some remarks on prediction of protein-protein interaction with machine learning. *Medicinal Chemistry*, 11(3):254–264, 2015.

[82] Stanley Fields and Ok-kyu Song. A novel genetic system to detect protein–protein interactions. *Nature*, 340(6230):245–246, 1989.

[83] Yuen Ho, Albrecht Gruhler, Adrian Heilbut, Gary D Bader, Lynda Moore, Sally-Lin Adams, Anna Millar, Paul Taylor, Keiryn Bennett, Kelly Boutilier, et al. Systematic identification of protein complexes in saccharomyces cerevisiae by mass spectrometry. *Nature*, 415(6868):180–183, 2002.

[84] David E Gordon, Gwendolyn M Jang, Mehdi Bouhaddou, Jiewei Xu, Kirsten Obernier, Kris M White, Matthew J O'Meara, Veronica V Rezelj, Jeffrey Z Guo, Danielle L Swaney, et al. A sars-cov-2 protein interaction map reveals targets for drug repurposing. *Nature*, pages 1–13, 2020.

[85] Christian von Mering, Roland Krause, Berend Snel, Michael Cornell, Stephen G. Oliver, Stanley Fields, and Peer Bork. Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417(6887):399–403, 2002.

[86] Tanlin Sun, Bo Zhou, Luhua Lai, and Jianfeng Pei. Sequence-based prediction of protein protein interaction using a deep-learning algorithm. *BMC bioinformatics*, 18(1):1–8, 2017.

[87] Asa Ben-Hur and William Stafford Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21(suppl_1):i38–i46, 2005.

[88] Lei Yang, Jun-Feng Xia, and Jie Gui. Prediction of protein-protein interactions from protein sequence using local descriptors. *Protein and Peptide Letters*, 17(9):1085–1090, 2010.

[89] Shawn M Gomez, William Stafford Noble, and Andrey Rzhetsky. Learning to predict protein–protein interactions from protein sequences. *Bioinformatics*, 19(15):1875–1881, 2003.

[90] Shawn Martin, Diana Roe, and Jean-Loup Faulon. Predicting protein–protein interactions using signature products. *Bioinformatics*, 21(2):218–226, 2005.

[91] Yanzhi Guo, Lezheng Yu, Zhining Wen, and Menglong Li. Using support vector machine combined with auto covariance to predict protein–protein interactions from protein sequences. *Nucleic acids research*, 36(9):3025–3030, 2008.

[92] Zhu-Hong You, Lin Zhu, Chun-Hou Zheng, Hong-Jie Yu, Su-Ping Deng, and Zhen Ji. Prediction of protein-protein interactions from amino acid sequences using a novel multi-scale continuous and discontinuous feature set. In *BMC bioinformatics*, volume 15, page S9. Springer, 2014.

[93] Tobias Hamp and Burkhard Rost. Evolutionary profiles improve protein–protein interaction prediction from sequence. *Bioinformatics*, 31(12):1945–1950, 2015.

[94] Xiaodi Yang, Shiping Yang, Qinmengge Li, Stefan Wuchty, and Ziding Zhang. Prediction of human-virus protein-protein interactions through a sequence embedding-based machine learning method. *Computational and structural biotechnology journal*, 18:153–161, 2020.

[95] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Xi Chen, John Canny, Pieter Abbeel, and Yun S Song. Evaluating protein transfer learning with tape. *arXiv preprint arXiv:1906.08230*, 2019.

[96] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909*, 2019.

[97] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[98] Emma Redhead and Timothy L Bailey. Discriminative motif discovery in dna and protein sequences using the deme algorithm. *BMC bioinformatics*, 8(1):385, 2007.

[99] Norman E Davey, Gilles Travé, and Toby J Gibson. How viruses hijack cell regulation. *Trends in biochemical sciences*, 36(3):159–169, 2011.

[100] Qian Cong, Ivan Anishchenko, Sergey Ovchinnikov, and David Baker. Protein interaction networks revealed by proteome coevolution. *Science*, 365(6449):185–189, 2019.

[101] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[102] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.

[103] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[104] Florencio Pazos and Alfonso Valencia. Similarity of phylogenetic trees as indicator of protein–protein interaction. *Protein engineering*, 14(9):609–614, 2001.

[105] Lei Wang, Hai-Feng Wang, San-Rong Liu, Xin Yan, and Ke-Jian Song. Predicting protein-protein interactions from matrix-based protein sequence using convolution neural network and feature-selective rotation forest. *Scientific reports*, 9(1):1–12, 2019.

[106] Somaye Hashemifar, Behnam Neyshabur, Aly A Khan, and Jinbo Xu. Predicting protein–protein interactions through sequence-based deep learning. *Bioinformatics*, 34(17):i802–i810, 2018.

[107] Florian Richoux, Charlène Servantie, Cynthia Borès, and Stéphane Téletchéa. Comparing two deep learning sequence-based models for protein-protein interaction prediction. *arXiv preprint arXiv:1901.06268*, 2019.

[108] Kalyani B Karunakaran, N Balakrishnan, and Madhavi K Ganapathiraju. Interactome of sars-cov-2/ncov19 modulated host proteins with computationally predicted ppis, 2020.

[109] Anne-Florence Bitbol. Inferring interaction partners from protein sequences using mutual information. *PLoS computational biology*, 14(11):e1006401, 2018.

[110] Oznur Tastan, Yanjun Qi, Jaime G Carbonell, and Judith Klein-Seetharaman. Prediction of interactions between hiv-1 and human proteins by information integration, 2009.

[111] Yanjun Qi, Oznur Tastan, Jaime G Carbonell, Judith Klein-Seetharaman, and Jason Weston. Semi-supervised multi-task learning for predicting interactions between hiv-1 and human proteins. *Bioinformatics*, 26(18):i645–i652, 2010.

[112] Guangyu Cui, Chao Fang, and Kyungsook Han. Prediction of protein-protein interactions between viruses and human by an svm model. In *BMC bioinformatics*, volume 13, page S5. Springer, 2012.

[113] Esmaeil Nourani, Farshad Khunjush, and Saliha Durmuş. Computational approaches for prediction of pathogen-host protein-protein interactions. *Frontiers in microbiology*, 6:94, 2015.

[114] Yungki Park and Edward M Marcotte. Flaws in evaluation schemes for pair-input computational predictions. *Nature methods*, 9(12):1134, 2012.

[115] Yanjun Qi, Merja Oja, Jason Weston, and William Stafford Noble. A unified multitask architecture for predicting local protein properties. *PloS one*, 7(3):e32235, 2012.

[116] Zeming Lin, Jack Lanchantin, and Yanjun Qi. Must-cnn: a multilayer shift-and-stitch deep convolutional architecture for sequence-based protein structure prediction. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[117] Seonwoo Min, Seunghyun Park, Siwon Kim, Hyun-Soo Choi, and Sungroh Yoon. Pre-training of deep bidirectional protein sequence representations with structural information, 2019.

[118] Tristan Bepler and Bonnie Berger. Learning protein sequence embeddings using information from structure. *arXiv preprint arXiv:1902.08661*, 2019.

[119] Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. Cloze-driven pretraining of self-attention networks. *arXiv preprint arXiv:1903.07785*, 2019.

[120] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[121] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853, 2005.

[122] Dekang Lin and Xiaoyun Wu. Phrase clustering for discriminative learning. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1030–1038. Association for Computational Linguistics, 2009.

[123] Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*, 2017.

[124] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning, 2016.

[125] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[126] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

[127] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

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

[129] UniProt Consortium. Uniprot: a worldwide hub of protein knowledge. *Nucleic acids research*, 47(D1):D506–D515, 2019.

[130] Michael Schantz Klausen, Martin Closter Jespersen, Henrik Nielsen, Kamilla Kjaergaard Jensen, Vanessa Isabell Jurtz, Casper Kaae Soenderby, Morten Otto Alexander Sommer, Ole Winther, Morten Nielsen, Bent Petersen, et al. Netsurfp-2.0: Improved prediction of protein structural features by integrated deep learning. *Proteins: Structure, Function, and Bioinformatics*, 87(6):520–527, 2019.

[131] Mohammed AlQuraishi. End-to-end differentiable learning of protein structure. *Cell systems*, 8(4):292–301, 2019.

[132] Jie Hou, Badri Adhikari, and Jianlin Cheng. Deepsf: deep convolutional neural network for mapping protein sequences to folds. *Bioinformatics*, 34(8):1295–1303, 2018.

[133] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[134] Alejandro A Schäffer, L Aravind, Thomas L Madden, Sergei Shavirin, John L Spouge, Yuri I Wolf, Eugene V Koonin, and Stephen F Altschul. Improving the accuracy of psi-blast protein database searches with composition-based statistics and other refinements. *Nucleic acids research*, 29(14):2994–3005, 2001.

[135] Michael Remmert, Andreas Biegert, Andreas Hauser, and Johannes Söding. Hhblits: lightning-fast iterative protein sequence searching by hmm-hmm alignment. *Nature methods*, 9(2):173, 2012.

[136] Kevin K Yang, Zachary Wu, and Frances H Arnold. Machine-learning-guided directed evolution for protein engineering. *Nature methods*, 16(8):687–694, 2019.

[137] Mais G Ammari, Cathy R Gresham, Fiona M McCarthy, and Bindu Nanduri. Hpidb 2.0: a curated database for host–pathogen interactions. *Database*, 2016, 2016.

[138] Rose Oughtred, Chris Stark, Bobby-Joe Breitkreutz, Jennifer Rust, Lorrie Boucher, Christie Chang, Nadine Kolas, Lara O'Donnell, Genie Leung, Rochelle McAdam, et al. The biogrid interaction database: 2019 update. *Nucleic acids research*, 47(D1):D529–D541, 2019.

[139] Tyler N Starr, Allison J Greaney, Sarah K Hilton, Daniel Ellis, Katharine HD Crawford, Adam S Dingens, Mary Jane Navarro, John E Bowen, M Alejandra Tortorici, Alexandra C Walls, et al. Deep mutational scanning of sars-cov-2 receptor binding domain reveals constraints on folding and ace2 binding. *Cell*, 182(5):1295–1310, 2020.

[140] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Kathryn Tunyasuvunakool, Olaf Ronneberger, Russ Bates, Augustin Žídek, Alex Bridgland, et al. High accuracy protein structure prediction using deep learning. *Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book)*, 22:24, 2020.

[141] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2285–2294, 2016.

[142] Zhouxia Wang, Tianshui Chen, Guanbin Li, Ruijia Xu, and Liang Lin. Multi-label image recognition by recurrently discovering attentional regions. In *Proceedings of the IEEE international conference on computer vision*, pages 464–472, 2017.

[143] Shang-Fu Chen, Yi-Chen Chen, Chih-Kuan Yeh, and Yu-Chiang Wang. Order-free rnn with visual attention for multi-label classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[144] Chung-Wei Lee, Wei Fang, Chih-Kuan Yeh, and Yu-Chiang Frank Wang. Multi-label zero-shot learning with structured knowledge graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1576–1585, 2018.

[145] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, and Yanwen Guo. Multi-Label Image Recognition with Graph Convolutional Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[146] Tianlu Wang, Kota Yamaguchi, and Vicente Ordonez. Feedback-prop: Convolutional neural network inference under partial evidence. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[147] Ming-Kun Xie and Sheng-Jun Huang. Partial multi-label learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[148] Thibaut Durand, Nazanin Mehrasa, and Greg Mori. Learning a deep convnet for multi-label classification with partial labels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 647–657, 2019.

[149] Kaustav Kundu and Joseph Tighe. Exploiting weakly supervised visual patterns to learn from partial annotations. *Advances in Neural Information Processing Systems*, 33, 2020.

[150] Wilson L Taylor. "cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

[151] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[152] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5338–5348. PMLR, 13–18 Jul 2020.

[153] Tianshui Chen, Muxin Xu, Xiaolu Hui, Hefeng Wu, and Liang Lin. Learning semantic-specific graph representation for multi-label image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 522–531, 2019.

[154] Tianshui Chen, Liang Lin, Xiaolu Hui, Riquan Chen, and Hefeng Wu. Knowledge-guided multi-label few-shot learning for general image recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[155] Feng Zhu, Hongsheng Li, Wanli Ouyang, Nenghai Yu, and Xiaogang Wang. Learning spatial regularization with image-level supervisions for multi-label image classification. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2027–2036, 2017.

[156] Weifeng Ge, Sibei Yang, and Yizhou Yu. Multi-evidence filtering and fusion for multi-label classification, object detection and semantic segmentation based on weakly supervised learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1277–1286, 2018.

[157] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[158] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123:32–73, 2016.

[159] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. *California Institute of Technology*, 2010.

[160] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[161] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[162] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 331–335, Florence, Italy, July 2019. Association for Computational Linguistics.

[163] Jia Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *European conference on computer vision*, pages 48–64. Springer, 2014.

[164] Vicente Ordonez, Wei Liu, Jia Deng, Yejin Choi, Alexander C Berg, and Tamara L Berg. Predicting entry-level categories. *International Journal of Computer Vision*, 115(1):29–43, 2015.

[165] Yong Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. Structure inference net: Object detection using scene-level context and instance-level relationships. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6985–6994, 2018.

[166] Hexiang Hu, Guang-Tong Zhou, Zhiwei Deng, Zicheng Liao, and Greg Mori. Learning structured inference neural networks with label relations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2960–2968, 2016.

[167] Pierre Stock and Moustapha Cisse. Convnets and imagenet beyond accuracy: Understanding mistakes and uncovering biases. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 498–512, 2018.

[168] Krzysztof Dembczynski, Weiwei Cheng, and Eyke Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML*, 2010.

[169] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning and Knowledge Discovery in Databases*, pages 254–269, 2009.

[170] Jinseok Nam, Eneldo Loza Mencía, Hyunwoo J Kim, and Johannes Fürnkranz. Maximizing subset accuracy with recurrent neural networks in multi-label classification. In *Advances in Neural Information Processing Systems*, pages 5419–5429, 2017.

[171] Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. Learning deep latent space for multi-label classification. In *AAAI*, pages 2838–2844, 2017.

[172] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pages 730–738, 2015.

[173] John Lafferty, Andrew McCallum, and Fernando CN Pereira. *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*. 2001.

[174] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 6(Sep):1453–1484, 2005.

[175] David Belanger and Andrew McCallum. Structured prediction energy networks. In *International Conference on Machine Learning*, pages 983–992, 2016.

[176] Yuhong Guo and Suicheng Gu. Multi-label classification using conditional dependency networks. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1300, 2011.

[177] Qiang Li, Maoying Qiao, Wei Bian, and Dacheng Tao. Conditional graphical lasso for multi-label image classification. In *CVPR*, pages 2977–2986, 06 2016.

[178] Xin Li, Feipeng Zhao, and Yuhong Guo. Multi-label image classification with a probabilistic label enhancement model. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14, pages 430–439, Arlington, Virginia, USA, 2014. AUAI Press.

[179] Mark Yatskar, Vicente Ordonez, Luke Zettlemoyer, and Ali Farhadi. Commonly uncommon: Semantic sparsity in situation recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7196–7205, 2017.

[180] Tejaswi Nimmagadda and Anima Anandkumar. Multi-object classification and unsupervised scene understanding using deep learning features and latent tree probabilistic models. *arXiv preprint arXiv:1505.00308*, 2015.

[181] Jack Lanchantin, Arshdeep Sekhon, and Yanjun Qi. Neural message passing for multi-label classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 138–163. Springer, 2019.

[182] Ziyan Yang, Leticia Pinto-Alva, Franck Dernoncourt, and Vicente Ordonez. Using visual feature space as a pivot across languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3673–3678, Online, November 2020. Association for Computational Linguistics.

[183] Yujia Huang, James Gornet, Sihui Dai, Zhiding Yu, Tan Nguyen, Doris Tsao, and Anima Anandkumar. Neural networks with recurrent generative feedback. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 535–545. Curran Associates, Inc., 2020.

[184] Michal Koperski, Tomasz Konopczynski, Rafal Nowak, Piotr Semberecki, and Tomasz Trzcinski. Plugin networks for inference under partial evidence. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2883–2891, 2020.

[185] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[186] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning. In *European Conference on Computer Vision*, pages 104–120. Springer, 2020.

[187] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018.

[188] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[189] Fuwen Tan, Jiangbo Yuan, and Vicente Ordonez. Instance-level image retrieval using reranking transformers. *arXiv preprint arXiv:2103.12236*, 2021.

[190] Hao Yang, Joey Tianyi Zhou, Yu Zhang, Bin-Bin Gao, Jianxin Wu, and Jianfei Cai. Exploit bounding box annotations for multi-label object recognition. In Lourdes Agapito, Tamara Berg, Jana Kosecka, and Lihi Zelnik-Manor, editors, *Proceedings - 29th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pages 280–288, United States of America, 2016. IEEE.

[191] Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan. Hcp: A flexible cnn framework for multi-label image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9):1901–1907, Sep. 2016.

[192] Zhouxia Wang, Tianshui Chen, Guanbin Li, Ruijia Xu, and Liang Lin. Multi-label image recognition by recurrently discovering attentional regions. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[193] Xiangyang Xue, Wei Zhang, Jie Zhang, Bin Wu, Jianping Fan, and Yao Lu. Correlative multi-label multi-instance image annotation. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 651–658, USA, 2011. IEEE Computer Society.

[194] Hongyu Su and Juho Rousu. Multilabel classification through random graph ensembles. In *Asian Conference on Machine Learning*, pages 404–418, 2013.

[195] Kazuyuki Hara, Daisuke Saitoh, and Hayaru Shouno. Analysis of dropout learning regarded as ensemble learning. In *International Conference on Artificial Neural Networks*. Springer, 2016.

[196] Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. *arXiv preprint arXiv:1707.09457*, 2017.

[197] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[198] John X Morris, Eli Lifland, Jack Lanchantin, Yangfeng Ji, and Yanjun Qi. Reevaluating adversarial examples in natural language. *arXiv preprint arXiv:2004.14174*, 2020.

[199] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 50–56. IEEE, 2018.

[200] Jennifer A Doudna and Emmanuelle Charpentier. The new frontier of genome engineering with crispr-cas9. *Science*, 346(6213), 2014.

[201] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. *arXiv preprint arXiv:1503.08895*, 2015.