THE UNIVERSITY OF CHICAGO

FEATURE TRANSFORMATIONS TO ENHANCE REPRESENTATION LEARNING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
RIDA ASSAF

CHICAGO, ILLINOIS
JUNE 2022

To Nadine, I like you

”The cure for boredom is curiosity. There is no cure for curiosity.” - Dorothy Parker

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

As is customary, I begin with special thanks to my friends and family. To my father Mahmoud, my sister Mariam, and my mother Najat (who wasted no chance to ask about my progress and push me forward), thank you. To my friends that I dare not list the names of lest one slips my mind, you know who you are, thank you.

I dedicate the remainder of this page to thank those whose support to me was not dictated by immediate biological or social factors. To the members of my committee: Rick Stevens, Fangfang Xia, and Ian Foster.

Thank you Ian for your continuous support and valuable insight. Your words of encouragement were among the first I received following my visa incident, and when I had to overcome research hurdles.

Fangfang, you have my sincerest thanks for always making the time to meet with me, especially after circumstances drove us apart and made it a challenge. Thank you for always knowing what to say and providing the most valuable suggestions. Thank you for bearing with my rants and supporting me throughout this journey.

Rick, I can't think of many people who would have stuck by me when it was easier not to. With so much on your plate and so much of the world depending on it, thank you for making me a part of your world and guiding me through this research journey.

I often wondered how learning in a Ph.D. happens. In my experience, after passing the required courses, the bulk of learning happened through work performed in solitary, with sporadic feedback and comments. It is through these comments that I found myself unlocking deeper levels of insight and understanding. Doing research felt like being launched into an unfamiliar territory, moving forward at various paces while being prodded in the right direction. Thank you for your constant prodding, and I hope to continue to learn from you while I'm welcome to.

ABSTRACT

Machine learning has crowned itself as a breakthrough in a number of domains, such as computer vision and natural language processing, matching and sometimes exceeding human level performance on certain supervised learning tasks. A major factor driving these success stories is the effort undertaken in designing different artificial neural network architectures that are particularly equipped to handle specialized tasks. For example, convolutional neural networks (CNNs) were designed to leverage spatial locality and other properties assumed in images, whereas recurrent neural networks and attention mechanisms are used with sequential data. While the majority of datasets available may not exhibit a special structure and is presented in plain tabular form, the most commonly used artificial neural network on tabular datasets is the multi-layer perceptron (MLP).

Similar to how choosing the right learner architecture may be considered as part of the training process that is essential for a good performance, we believe that some effort on the data representation side could further improve the learning outcome. The underlying idea when using artificial neural networks is that representation learning happens automatically using raw data, unlike classical machine learning methods that are still widely applied on tabular datasets, and that are often accompanied by feature selection and transformation techniques. We hypothesize that certain feature transformations could enhance the representation learning process and the subsequent machine learning outcome achieved by artificial neural networks, and propose three feature transformation techniques. The key principles underlying these transformations are highlighting the entities and relationships described by the data. The first transformation leverages our domain knowledge by manually designing visual representations of the features to be used by a two-dimensional CNN. Within the tabular space, one transformation generates a partitioning of the input feature vector such that each partition represents an entity or a relationship, to be used by a modular MLP (an MLP with multiple input layers). Another transformation generates a permutation of the feature vector where related features are neighbors, to be used with a one-dimensional CNN to capture the implicit feature groups. We provide empirical evidence suggesting that such transformations yield better results than baseline MLPs and existing methods, and require less time, data, and parameters.

We propose a method to automate the transformation process, and evaluate it empirically using synthetic and real-world datasets. The synthetic datasets are designed in a way that allows different levels of representation of the underlying entities and relationships, providing additional insight into the learning process. The real-world datasets are derived from experiments reported by other approaches that propose automatic feature transformation techniques to enhance deep learning performance. Our results show a clear advantage over these approaches, not only in the learning outcome and the resources required, but also in the simplicity of method.

CHAPTER 1
INTRODUCTION

1.1 Motivation

To a human, the world is made up of entities perceived through the senses and symbolized through internal mental processes. These entities are defined using the features and attributes that describe them, and the relationships between them. Much of learning involves discovering these features and relationships. The input and output to such processes make up information. To a machine, that information can be presented in different forms. One of these forms is referred to as tabular, that is a textual representation comprised of categorical and numerical values. Another form of representation is visual, which is comprised of images including shapes that are familiar to the human vision system.

In [1], Goyal A. and Bengio Y. present inductive biases as ”training data in disguise”, helping machine learning models exploit our a priori knowledge of the problem by having artificial neural network architectures that make use of the properties associated with these biases. For example, convolutional neural networks capture group equivariance and spatial locality, recurrent neural networks exploit equivariance over time and capture sequential dependency, graph neural networks exploit equivariance over entities and relationships, and deep architectures learn complicated functions via the composition of simpler ones. These neural networks achieve state of the art performance in their respective domains, because they leverage prior knowledge over the corresponding domain data. Thus, the choice of learner is essential for the success of the learning process. We believe that similar to how prior knowledge guides our decisions in designing and using these learner architectures, similar consideration regarding how the data is represented to the learner could also improve the learning performance.

Much work has been done on feature engineering and feature selection. The outcome of feature selection methods is a subset of the features presumed to be strong predictors, that are then used as input to the learner. Supervised feature selection methods include the target variable in the process of determining the strong predictors, contrary to unsupervised feature selection methods. One class of feature selection methods is known as wrapper methods, that works by evaluating multiple machine learning models using different combinations of features, and selecting the combination that maximizes the learning performance. Another class known as the filter feature selection methods statistically evaluates the relationship between input features and the target variable and chooses a subset of the features that score highest [2]. In these approaches, the features that do not make it to the final cut are discarded from the downstream learning process. While some models are intrinsically resistant to non-informative features (e.g. ensemble tree based methods), classical feature selection techniques are historically used with classical machine learning algorithms. When it comes to deep learning, the idea is that the model would automatically learn the appropriate representation without the need for feature selection and pre-processing. We show that feature selection and feature transformation techniques may facilitate the representation learning process done by neural networks.

Generally, the existing approaches presented in this work attempt to find groups of features that serve as strong predictors, without discarding any features. Some approaches work by mapping tabular features into the visual space using distance metrics that result in related features being represented as neighboring pixels, resulting in different groups of features being picked up by kernels in a convolutional neural networks. The pre-processing steps used to measure the feature relationships and apply the subsequent mappings are done on a global level. Other approaches apply more local methods e.g. by applying the feature selection per learning instance.

We propose new feature transformation strategies that aim to highlight the entities and relationships represented in the input feature vector. One way to do that is by leveraging human domain knowledge and representing the underlying information visually through images with distinct shapes and colors similar to what a human expert would use to make judgements on similar tasks. We hypothesize that a primary difference in visual learning is that objects are presented separately (provided appropriate kernel sizes) to the learner. Whereas tabularly, the learner may first need to learn what each entity is formed of (which subsets of features) and then what the relationships between those entities are. Such a representation may further make use of the pre-trained neural networks that are powering the advances in computer vision applications and are already adept at handling low-level visual features.

We demonstrate the effectiveness of direct visual representation of tabular features on two problems from the bioinformatics domain, by representing genomic data visually in a way that highlights genes and relationships between them. In these problems, an additional benefit of our approach comes from representing certain features that are not easily represented tabularly (gene conservation, which lends itself to nice spatial representation) more naturally across the two-dimensional space made possible by using images.

In cases where the information involves entities and relationships clearly represented by different feature groups, our approach tweaks the machine learning model's architecture while maintaining a tabular feature representation. Namely, we resort to a modular learning approach, where the input feature vector is partitioned into groups that are fed independently through multiple input layers to a multi-layer perceptron. The idea of partitioning the feature vector stems from our conjecture that not all features will be informative for every output prediction, and that non-informative features may act as noise. We hypothesize that in some cases, a learner that does not exploit feature groups may perform poorly compared to one that does, or may need more time or parameters to compensate. In cases where knowledge based feature groups are lacking, we use the feature correlation matrix to infer potential feature groups. If feature groups are not easily defined, our approach re-orders the features based on their correlation matrix, and uses a one-dimensional convolutional neural network as a learner that can exploit the generated spatial locality.

We evaluate our approach empirically using synthetic datasets that allow for different representations of entities and the relationships between them. We also compare our approach to existing ones using real-world datasets. We demonstrate that our approach performs better than the other methods and shows an advantage in terms of the number of parameters and training time required on some of the machine learning tasks.

1.2 Contributions

Our first contribution is showing that feature transformations highlighting the entities and relationships described by the data facilitate the process of representation learning and yield a better learning outcome. Our second contribution is a state of the art method for identifying genomic islands in bacterial genomes. Our third contribution is a state of the art method for detecting operons in bacterial genomes. These methods are applications of our proposed feature transformation strategies. Our fourth contribution is proposing general tabular feature transformation techniques, and a pipeline to automate them. Our fifth contribution is providing empirical evidence using synthetic and real world datasets, and drawing insight into the learning process by comparing the tabular and the visual modes of learning. We do that by training different learners on the same information represented with different levels of entity and relationship highlighting. Our sixth contribution is critiquing existing methods attempting to automate feature engineering and transformation, and showing that the results reported by many studies published in the literature could be outperformed not only by our method, but also by simpler machine learning models.

1.3 Dissertation Organization

Even with the constantly increasing sequenced genomic data, having meaningful and labeled datasets is still a challenge, which makes supervised learning applications on some bioinformatics problems more infeasible. There are numerous problems that suffer from limited datasets in bioinformatics. In chapters 2 and 3, we transform two of these problems in to computer vision tasks, by representing the data visually in a way that captures the most prominent features according to human experts, and highlights the relevant entities and relationships. The results we achieve are better than those reported by state of the art methods. The two problems we present are identifying genomic islands, and operons, in bacterial genomes. These are two well-known genome annotation problems, known for being experimentally costly and to suffer from the lack of extensive verified datasets. In chapter 4, we present an approach to transform data in a way that highlights the entities and relationships while maintaining a tabular data representation, and propose a method to automate this feature transformation process. In chapter 5, we use synthetic datasets to empirically evaluate our proposed method and examine the learning process of visual and tabular machine learners. In chapter 6, we compare our approach to others that propose automatic feature selection and transformation techniques, using real-world datasets. Finally in chapter 7, we present a summary of our work and outline some future directions.

CHAPTER 2
IDENTIFYING GENOMIC ISLANDS VIA VISUAL REPRESENTATION LEARNING

In this chapter, we present an application of feature transformation on genomic data. We represent genomic fragments visually in a way designed to leverage our domain knowledge of the underlying features, and use these visual representations to identify genomic islands (GIs) in bacterial genomes. This problem is known to suffer from extremely limited ground-truth datasets. Our method (Shutter Island) [3] uses deep neural networks previously trained on computer vision tasks, and achieves a superior performance in capturing the union of the predictions made by other tools, in addition to making novel predictions that exhibit GI features.

2.1 Introduction

2.1.1 Background

There are a number of ways through which bacteria may evolve, including deletion events, horizontal gene transfer, and single-point mutations, as can be seen in Figure 2.1 [4].

Horizontal gene transfer is the main source of adaptability for bacteria, through which genes are obtained from different sources including bacteria, archaea, viruses, and eukaryotes. This process promotes the rapid spread of genetic information across lineages, typically in the form of clusters of genes referred to as genomic islands. Different types of GIs exist, and are often classified by the content of their cargo genes or their means of integration and mobility (Figure 2.2 [5]).

In the 1990s, some Escherichia coli strains were found to have exclusive virulence genes that were not found in other strains [6, 7]. These genes were thought to have been acquired horizontally and were referred to as pathogenicity islands (PAIs). Further investigations

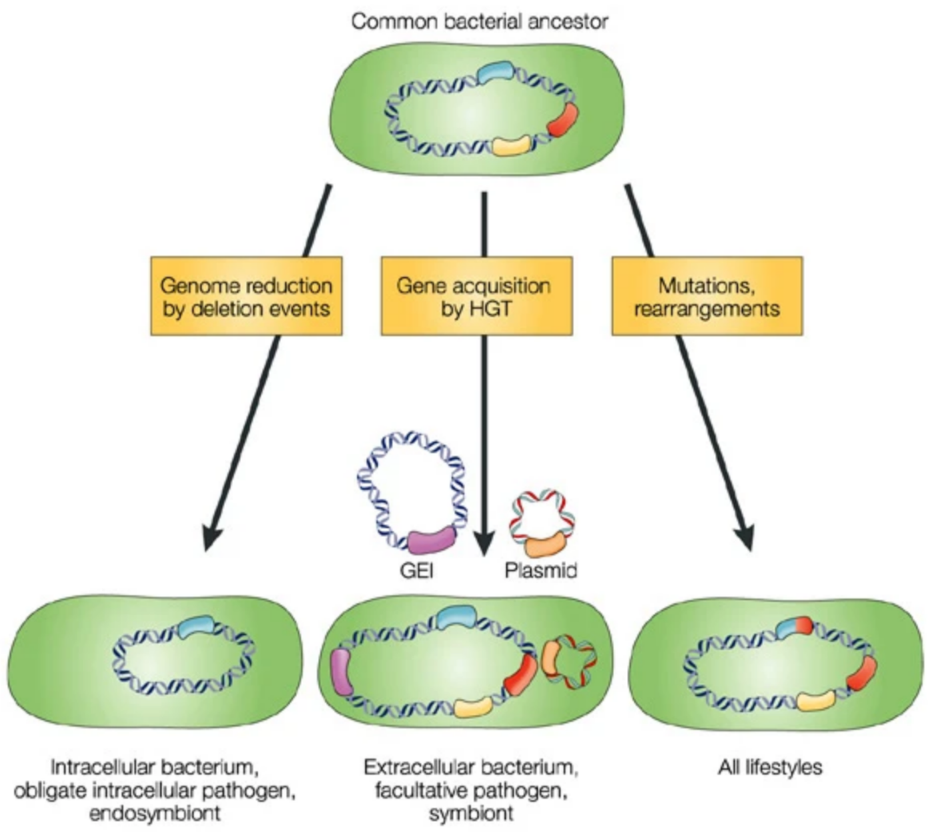

Common bacterial ancestor
Genome reduction by deletion events
Gene acquisition by HGT
Mutations, rearrangements
GEI
Plasmid
Intracellular bacterium, obligate intracellular pathogen, endosymbiont
Extracellular bacterium, facultative pathogen, symbiont
All lifestyles

Figure 2.1: Different mechanisms of bacterial evolution: genome reduction by deletion events, gene acquisition by horizontal gene transfer, mutations and rearrangements. Adapted from (U. Dobrindt et. al. 2004 [4]).

showed that other types of islands carrying other types of genes exist, giving rise to more names such as ”secretion islands,” ”resistance islands,” and ”metabolic islands,” since the genes carried by these islands could promote not only virulence but also symbiosis or catabolic pathways [4, 8, 9].

Aside from functionality, different names are also assigned to islands on the basis of their mobility. Some GIs are mobile and can thus move themselves to new hosts, such as conjugative transposons, integrative and conjugative elements (ICEs), and prophages (Figure 2.3 [6]), whereas other GIs lose their mobility [10, 11]. Prophages are viruses that infect bacteria and then remain inside the cell and replicate with the genome [12]. They are also referred to as bacteriophages, and constitute the majority of viruses, outnumbering bacteria by a factor of ten to one [13, 14]. A genomic island then is a cluster of genes that is typically between 10 kb and 200 kb in length and has been transferred horizontally [15].

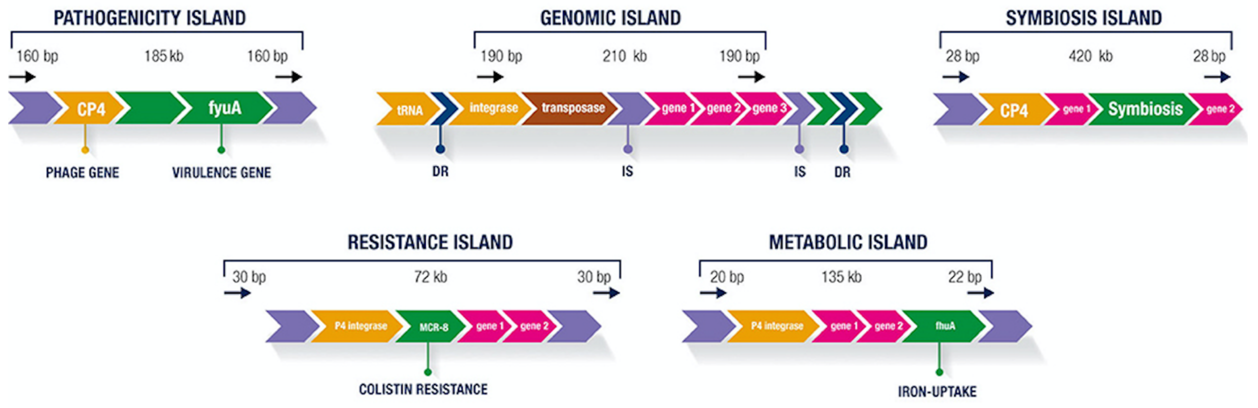


Figure 2.2: Different types of genomic islands, as characterized by the genes they carry and the functions they promote. Adapted from (Antonio Camilo da Silva Filho et. al. 2018 [5]).

Horizontal gene transfer (HGT) may contribute to anywhere between 1.6% and 32.6% of a bacterial genome [16]. This percentage implies that a major factor in the variability across bacterial species and clades can be attributed to GIs [17]. Thus, GIs impose an additional challenge to our ability to reconstruct the evolutionary tree of life. The identification of GIs is also important for the advancement of medicine, by helping develop new vaccines and

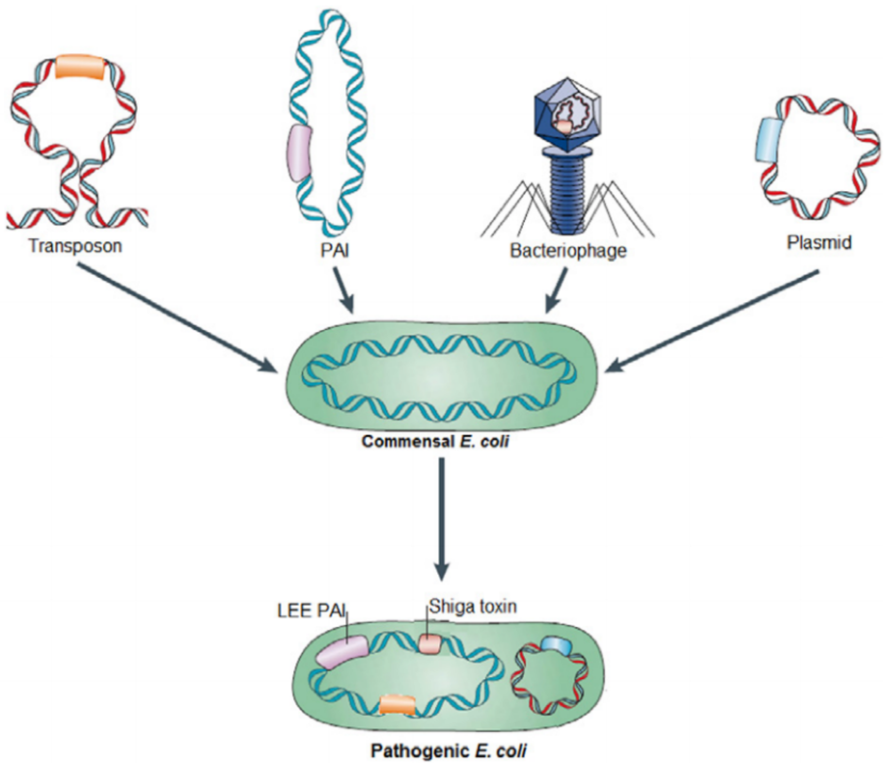


Figure 2.3: A visualization example of how different types of genomic islands (transposon, pathogenicity island, bacteriophage, plasmid) may change a commensal E. coli into a pathogenic one. Adapted from (Morgan G. I. Langille 2010 [6]).

antibiotics [18] or cancer therapies [19]. For example, knowing that PAIs can carry many pathogenicity genes and virulence genes [20, 21, 22], researchers found that potential vaccine candidates resided within them [23].

We propose that the problem of predicting genomic islands computationally is an excellent candidate for transfer learning on visual representations that are engineered to highlight the entities (genes) and relationships (e.g. functional relatedness, conservation) between them. Visual representation learning also alleviates the problem of the extreme limitation of available ground-truth datasets and enables the use of powerful deep learning technologies. We present a method (Shutter Island) that uses deep neural networks, previously trained on computer vision tasks, for the detection of genomic islands. Using a manually verified reference dataset, Shutter Island proved to be superior to the existing tools in generalizing over the union of their predicted results. Moreover, Shutter Island makes novel predictions that show GI features.

2.1.2 Related Work

Methods proposed for the prediction of GIs fall under two categories: those that rely on sequence composition analysis and those that rely on comparative genomics. We present an overview of some of these methods next.

Islander works by first identifying tRNA genes and their fragments as endpoints to candidate regions, then disqualifying candidates through a set of filters such as sequence length and the absence of an integrase gene [8]. *IslandPick* identifies GIs by comparing the query genome with a set of related genomes selected by an evolutionary distance function [24]. It uses Blast and Mauve for the genome alignment. The outcome heavily depends on the choice of reference genomes selected. *Phaster* uses BLAST against a phage-specific sequence database (the NCBI phage database and the database developed by Srividhya et al. [25]), followed by DBSCAN [26] to cluster the hits into prophage regions. *IslandPath-DIMOB* considers a genomic fragment to be an island if it contains at least one mobility gene, in addition to 8 or more consecutive open reading frames with dinucleotide bias [27]. *SIGI-HMM* uses the Viterbi algorithm to analyze each gene's most probable codon usage states, comparing it against codon tables representing microbial donors or highly expressed genes, and classifying it as native or non-native accordingly [28]. *PAI-IDA* uses the sequence composition features, namely, GC content, codon usage, and dinucleotide frequency, to detect GIs [29]. *Alien Hunter* uses k-mers of variable length to perform its analysis, assigning more weight to longer k-mers [30]. *Phispy* uses random forests to classify windows based on features that include transcription strand directionality, customized AT and GC skew, protein length, and abundance of phage words [12]. *Phage Finder* classifies 10 kb windows with more than 3 bacteriophage-related proteins as GIs [31]. *IslandViewer* is an ensemble method that combines the results of three other tools—SIGI-HMM, IslandPath-DIMOB, and IslandPick—into one web resource [32].

2.1.3 Challenges

No single tool is able to detect all GIs in all bacterial genomes. Methods that narrow their search to GIs that integrate under certain conditions, such as into tRNAs, miss out on the other GIs. Similarly, not all GI regions exhibit atypical nucleotide content [33]. Evolutionary events such as gene loss and genomic rearrangement [4] present more challenges. For example, the presence of highly expressed genes or having closely related island host and donor might lead to false negatives [18]. Tools that use windows face difficulty in adjusting their size: small sizes lead to large statistical fluctuation, whereas larger sizes result in low resolution [34].

For comparative genomics methods, the outcomes depend strongly on the choice of genomes used in the alignment process. Very distant genomes may lead to false positives, and very close genomes may lead to false negatives. In general, the number of reported GIs may differ across tools, because one large GI is often reported as a few smaller ones or vice versa, making it harder to detect end points and boundaries accurately. The lack of experimentally verified ground-truth data-sets spanning the different types of GIs makes point-to-point comparison across the tools extremely challenging. Moreover, different tools follow different custom-defined metrics to judge their results, typically by using a threshold representing the minimum values of features (e.g., number of phage words) present in a region to be considered a GI, which adds to the complications of validating GI predictions and comparing tools' performances.

2.2 Results

No reliable GI dataset exists that can validate the predictions of computational methods [30]. Although several databases exist, they usually cover only specific types of GIs (e.g.

Islander, PAIDB, ICEberg), which would flag as false positives any extra predictions made by those tools. Moreover, as Nelson et al. state, "The reliability of the databases has not been verified by any convincing biological evidence" [10]. We validate the quality of the predictions made by our method first by using metrics reported in previous studies, then by introducing novel metrics and presenting some qualitative assessments of the predictions.

In Table 2.1, we present the total number of GI predictions made by each tool over the entire testing dataset, which consists of 34 genomes and is described in more detail in the Methods section.

Table 2.1: The number of islands and their total base pair count predicted by each tool over the testing genomes dataset.

Tool	Number of Islands	Number of Base Pairs
ShutterIsland	649	10,700,492
AlienHunter	1919	19,561,593
IslandViewer	701	10,571,974
IslandPath-Dimob	339	6,871,312
Phaster	109	4,334,225
Phispy	96	3,979,173
PhageFinder	85	3,656,950
IslandPick	362	3,020,733
SIGI-HMM	359	2,543,145
Islander	50	2,019,610

We can see from Table 2.1 that Alien Hunter calls the most GIs, with almost double the amount called by Shutter Island and IslandViewer if measured by base pair count, and even more if measured by island count. However, as mentioned earlier, one island predicted by one tool could be predicted as several islands by another, partly due to the different length filters the tools apply. The number of predictions made by Shutter Island is close to that made by IslandViewer, which is an ensemble method combining four other tools' predictions.

2.2.1 Validation Following Previously Accepted Methods

In this section, we present a comparison of the tools' performance following definitions accepted by the scientific community and accepted as part of an earlier study introducing

Phispy [12]. To distinguish the results presented in this section, we use *Phispy* as a pre-fix to the names of the metrics used. Namely, We refer to the metrics used as *Phispy True Positives* (PTP), *Phispy False Positives* (PFP), and *Phispy False Negatives* (PFN). Note that Phispy did not define true negatives. A true positive can be verified by the presence of phage-related genes, and a false positive by their absence. But while a region that exhibits GI features but is not predicted as a GI can be defined as a false negative, regions not showing any GI features cannot be labeled as true negatives, due to our limited understanding of GI features. Even the task of deciding the region size would not be trivial.

Table 2.2 was constructed with the following definitions:

- A Phispy True Positive is a region predicted as a GI and:
  - Contains a phage-related gene, or
  - With at least 50% of its genes having unknown functions.
- A Phispy False Positive is a region predicted as a GI but does not satisfy the above conditions.
- A Phispy False Negative is a region with six consecutive phage-related genes that is not predicted as a GI.

We followed a similar approach as the one used by Phaster to determine the presence of phage-related genes, which is looking for certain keywords present in the genes' annotations. Throughout the remainder of the paper, we refer to genes with annotations that contain such keywords as GI features.

Using the Phispy metrics defined earlier, we present the true positive rate (sensitivity) and the percentage of false positive predictions in Table 2.2.

Note that While some tools report 0 Phispy False Positives, they also score significantly lower on the true positive rate metric, the reason being that these tools make much fewer predictions in general.

Table 2.2: True positive rate (Sensitivity) and the percentage of Phispy False Positives, as defined in the Phispy study, for predictions made by each tool over the entire testing dataset, comprised of 34 genomes.

Tool	Sensitivity (%)	False positives (%)
ShutterIsland	92.4	29
AlienHunter	91.8	50.7
IslandViewer	88.2	30
IslandPath-Dimob	80.9	2.7
Phaster	73.2	0
Phispy	68.6	0
PhageFinder	65.9	0
IslandPick	62.5	44.8
SIGI-HMM	66.4	30.6
Islander	31.8	0

2.2.2 Validation Using Novel Metrics

In this section, we present more general metrics to perform a more objective cross-tool comparison. Since every tool predicts a subset of all GIs, we capture the coverage of each tool across other tools' predictions in Table 2.3. We omit the tools we were not able to run, and use the default parameters for all the listed tools.

Table 2.3 shows that Alien Hunter's predictions overlap the most with those made by other tools, which is expected given that it has the highest base-pair coverage. Shutter Island comes next and overlaps the most with three of the presented tools' predictions. Note that while Shutter Island was trained only on the intersection of the predictions made by Phispy and IslandViewer, it generalizes and scores the highest overlap with predictions made by Phage Finder and Phaster.

Since some tools make many more predictions than do others, we used the GI features mentioned earlier to get a better idea about the quality of these overlapping predictions. In Table 2.4, we present the percentage of overlapping predictions that show GI features, followed by the percentage of non-overlapping predictions showing GI features. Tools that use these features to perform their classifications were omitted. We can see that on average, Shutter Island's overlapping predictions include GI features the most. Shutter Island also misses the least predictions made by other tools that show GI features. Finally, Shutter Island has the most predictions showing GI features that are not being predicted by other tools. Table 2.5 shows each tool's novel predictions that do not overlap with any of other tools', in addition to the percentage of those predictions with GI features. Alien Hunter's unique predictions almost outnumber every other tool's total predictions, and average 8 kilo base pairs (kbp) in length. Shutter Island's unique predictions have an average length of 14 kbp. Applying the same length cutoff threshold (8 kbp) on Alien Hunter's unique predictions reduces them to 301 islands with a total of 3,880,000 bp, which is on par with those made by Shutter Island's. However, a larger percentage of unique predictions made by Shutter Island exhibit GI features.

Next, we show the receiver operating characteristic (ROC) curve of our classifier in Figure 2.4. The construction of a ROC curve requires a definition of true negative predictions. Since our classifier performs its predictions on every gene in a genome, we consider the four genes flanking each side of every query gene, and introduce the following definitions. A region is:

- A true positive, if predicted as a GI and:
  - Includes a phage-related gene, or
  - Overlaps with a prediction made by another tool.
- A false positive, if predicted as a GI but does not satisfy the above conditions.
- A true negative, if not predicted as a GI and does not include a phage-related gene.
- A false negative , if not predicted as a GI but includes a phage-related gene.

Table 2.3: Cross-tool comparison of GI results: The percentage of GIs predicted over the testing dataset *Target*, that overlap with predictions made by other tools (Predictor).

Target / Predictor	ShutterIsland	IslandViewer	Phispy	PhageFinder	Islander	Phaster	Alien Hunter	IslandPick	Dimob	SIGI
ShutterIsland	N/A	45.7%	97.8%	99.1%	67.4%	92.9%	27%	20.3%	54%	28.8%
IslandViewer	42.8%	N/A	89.3%	89.2%	N/A	82.1%	39.4%	N/A	N/A	N/A
Phispy	29.1%	23.7%	N/A	98.3%	52.8%	79.3%	9%	10.8%	29.1%	11.5%
PhageFinder	28.1%	23.6%	92.8%	N/A	50.4%	79.8%	9%	10.1%	29/3%	12%
Islander	9.2%	21.2%	23.7%	25.7%	N/A	22.2%	8.3%	15.5%	22.9%	17%
Phaster	26.4%	22.5%	82.4%	86%	44.5%	N/A	10.4%	11.3%	27.5%	12.7%
AlienHunter	56.8%	78.9%	87.2%	86.5%	98%	87.2%	N/A	67.1%	82.8%	92.6%
IslandPick	10.6%	43.3%	25.4%	28.7%	51.7%	29%	13.8%	N/A	28.4%	31.2%
Dimob	34.9%	70.5%	86.1%	85.2%	87.8%	76.9%	25.7%	29.5%	N/A	50.3%
SIGI	17.2%	47.7%	34.4%	31.6%	63.2%	27.8%	22.6%	30.9%	44.8%	N/A

Table 2.4: Quality of overlapping predictions: The percentage of GIs predicted over the testing dataset by the *Target* tool, that overlap with predictions made by other tools (Predictor), that include GI features — The percentage of predictions made by the *Target* tool but not the Predictor, that include GI features.

Target / Predictor	ShutterIsland	Island Viewer	Alien Hunter	Island Pick	SIGI	**Average**
ShutterIsland	N/A	91% — 64%	87% — 47%	89% — 31%	87% — 36%	**89% — 45%**
IslandViewer	94% — 67%	N/A	89% — 45%	80% — n/a	87% — n/a	88% — 56%
AlienHunter	74% — 70%	66% — 60%	N/A	73% — 21%	71% — 42%	71% — 48%
IslandPick	69% — 76%	34% — 86%	49% — 53%	N/A	54% — 44%	52% — 65%
SIGI	67% — 75%	45% — 77%	48% — 51%	50% — 35%	N/A	53% — 60%

Table 2.5: The total number and base-pair count of unique predictions made by each tool over the testing genomes dataset, and the percentage of those predictions showing GI features.

Tool	Unique GIs (Count)	Unique GIs (Base pairs)	Unique GIs (GI features)
ShutterIsland	280	3,647,377	65%
AlienHunter	1155	9,583,497	40%
Phaster	2	30,814	0%
Phispy	1	26,890	100%

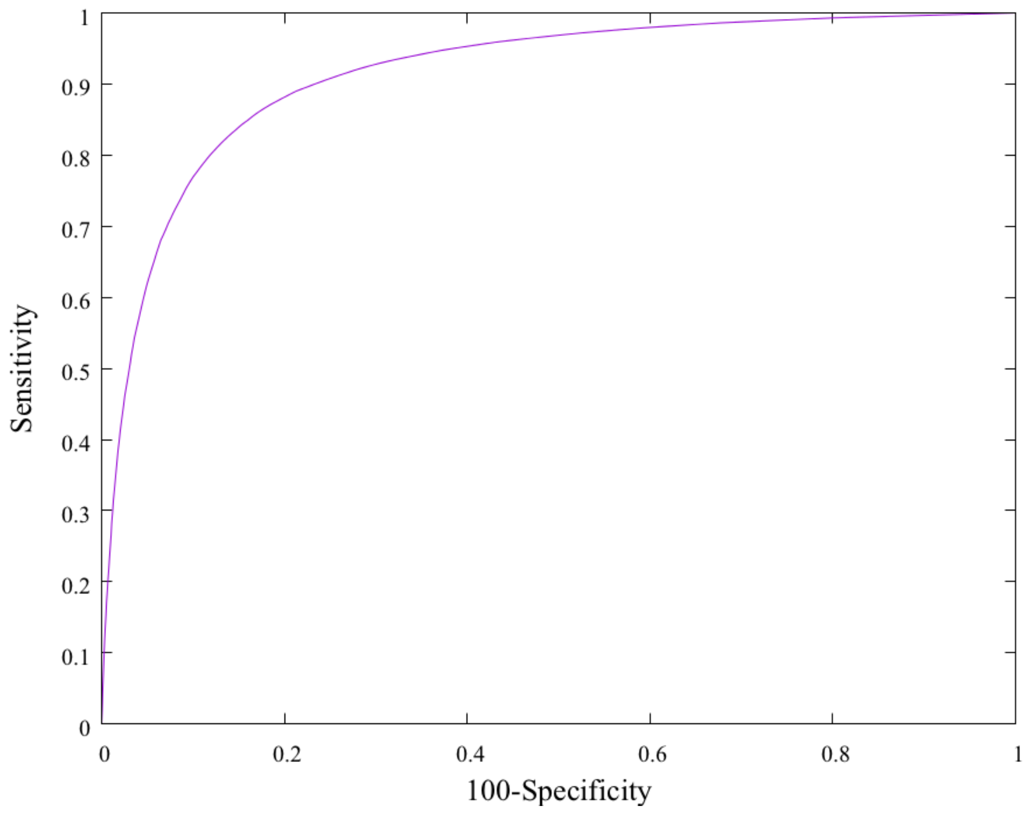
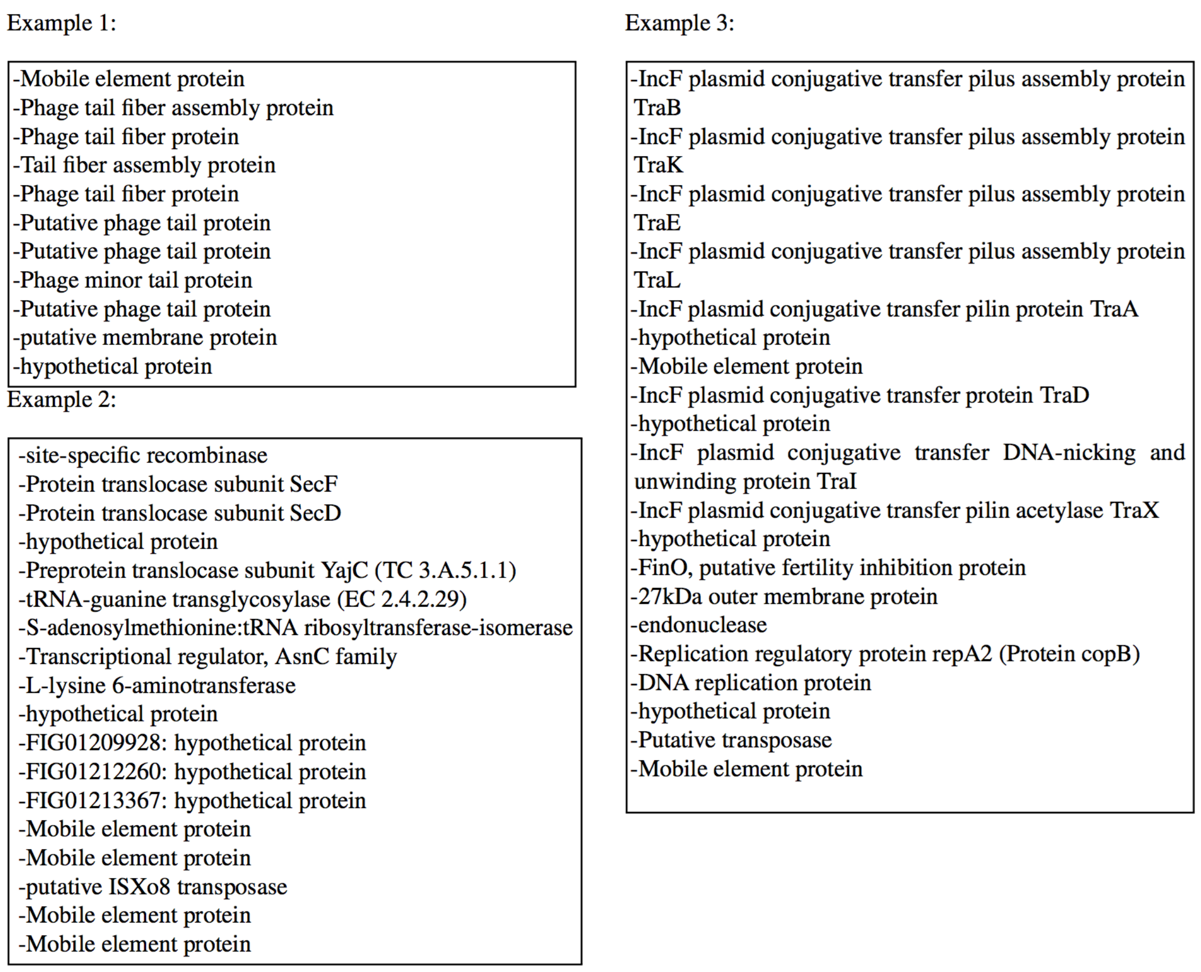


Figure 2.4: Receiver operating characteristic (ROC) curve for our genomic island binary classifier. The ROC curve plots the true positive rate as a function of the false positive rate. The greater the area under the curve is (the closer it is to the ideal top left corner point), the better.

2.2.3 Qualitative Assessment

To qualitatively assess the unique predictions made by Shutter Island, we present snapshots of the cargo genes typically found in these predicted regions in Figure 2.5, which shows that a significant number of the included genes carry GI related annotations.



Example 1:

-Mobile element protein
-Phage tail fiber assembly protein
-Phage tail fiber protein
-Tail fiber assembly protein
-Phage tail fiber protein
-Putative phage tail protein
-Putative phage tail protein
-Phage minor tail protein
-Putative phage tail protein
-putative membrane protein
-hypothetical protein

Example 2:

-site-specific recombinase
-Protein translocase subunit SecF
-Protein translocase subunit SecD
-hypothetical protein
-Preprotein translocase subunit YajC (TC 3.A.5.1.1)
-tRNA-guanine transglycosylase (EC 2.4.2.29)
-S-adenosylmethionine:tRNA ribosyltransferase-isomerase
-Transcriptional regulator, AsnC family
-L-lysine 6-aminotransferase
-hypothetical protein
-FIG01209928: hypothetical protein
-FIG01212260: hypothetical protein
-FIG01213367: hypothetical protein
-Mobile element protein
-Mobile element protein
-putative ISXo8 transposase
-Mobile element protein
-Mobile element protein

Example 3:

-IncF plasmid conjugative transfer pilus assembly protein TraB
-IncF plasmid conjugative transfer pilus assembly protein TraK
-IncF plasmid conjugative transfer pilus assembly protein TraE
-IncF plasmid conjugative transfer pilus assembly protein TraL
-IncF plasmid conjugative transfer pilin protein TraA
-hypothetical protein
-Mobile element protein
-IncF plasmid conjugative transfer protein TraD
-hypothetical protein
-IncF plasmid conjugative transfer DNA-nicking and unwinding protein TraI
-IncF plasmid conjugative transfer pilin acetylase TraX
-hypothetical protein
-FinO, putative fertility inhibition protein
-27kDa outer membrane protein
-endonuclease
-Replication regulatory protein repA2 (Protein copB)
-DNA replication protein
-hypothetical protein
-Putative transposase
-Mobile element protein

Figure 2.5: Examples of regions uniquely predicted by Shutter Island.

We also analyzed the most common gene annotations found in the unique predictions made by Shutter Island and Alien Hunter. We focused on these tools since they are the ones with a significant number of unique predictions to perform the analysis on. We observed that the most frequent genes that are common to these predicted regions are either of unknown functionality or are GI-related, which added to our confidence in these predictions. Specifically, 90% of genes uniquely predicted by Shutter Island are either of unknown function or labeled as a mobile element protein, compared to 64% of genes with similar labels uniquely predicted by Alien Hunter.

2.3 Methods

2.3.1 Datasets

PATRIC (the Pathosystems Resource Integration Center) is a bacterial bioinformatics resource center that we are part of (https://www.patricbrc.org) [35]. It provides researchers with the tools necessary to analyze their private data and to compare it with public data. PATRIC recently surpassed the 200,000 publicly sequenced genomes mark, ensuring that enough genomes are available for effective comparative genomics studies. For our training data, we used the set of reference+representative genomes found on PATRIC. For each genome, our program produced an image for every non-overlapping 10 kbp window. A balanced dataset was then curated from the total set of images created. Since this is a supervised learning approach and our goal is to generalize over the tools' predictions and beyond, we used Phispy and IslandViewer's predictions to label the images that belong to candidate islands. IslandViewer captures the predictions of different methods, and Phispy captures different GI features. To increase our confidence in the generated labels, we labeled a genomic fragment as a GI only if it was predicted as a GI by both of these tools.

To make predictions over novel genomes, our method generates an image for every gene in the genome. Each image is then classified as either part of a GI or not. This process generates a label for every gene in the genome. A length filter of 8 kbp is then applied, so that every group of genes labeled as part of a GI that spans more than 8 kbp is reported as a single GI.

Since no reliable benchmark is available, we used the set of genomes mentioned in Phispy to test our classifier. The set consists of 41 bacterial genomes, and the authors of Phispy reported that the GIs in these genomes have been manually verified [12]. Some of the tools used in the comparison have not been updated for a while, but most of the tools had predictions made over the genomes in this testing set. We discarded the genomes for which not all the tools reported predictions over, or that were part of the training set used to train our classifier, and ended up with a total of 34 genomes.

2.3.2 Feature Encoding Using Images

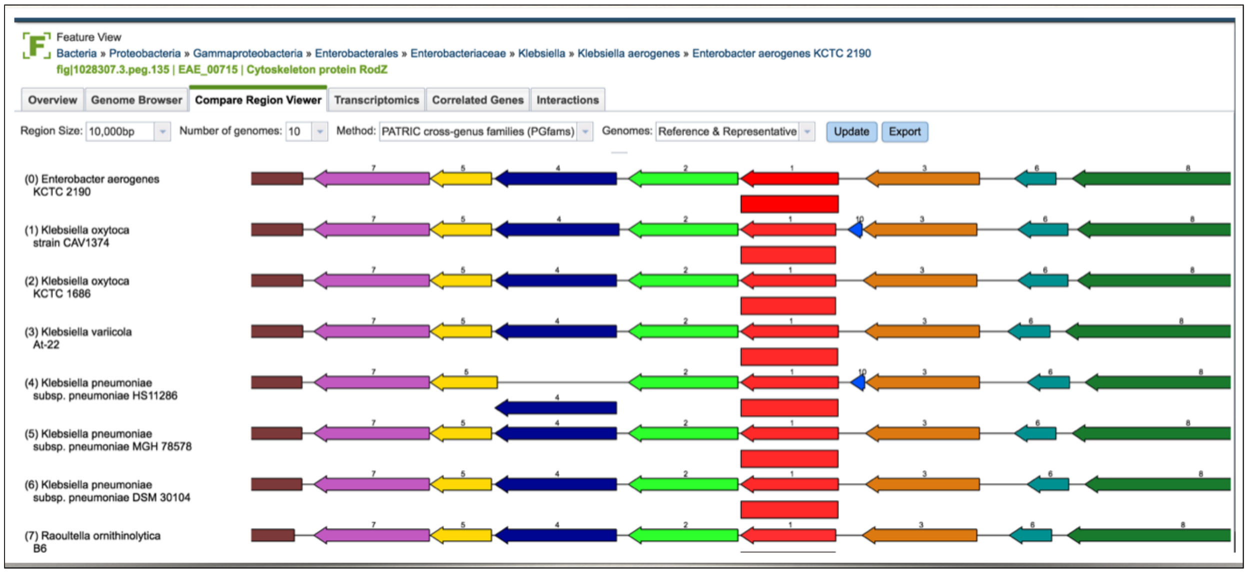
We present some of the most prominent features of genomic islands, listed by decreasing order of importance: [6, 36].

- One of the most important features of GIs is that they are sporadically distributed; that is, they are found only in certain isolates from a given strain or species.
- Since GIs are transferred horizontally across lineages and since different bacterial lineages have different sequence compositions, measures such as GC content or, more generally, oligonucleotides of various lengths (usually 2–9 nucleotides) are used [30, 37, 38]. Codon usage is a well-known metric, which is the special case of oligonucleotides of length 3.
- Since the probability of having outlying measurements decreases as the size of the region increases, tools usually use cut-off values for the minimum size of a region (or gene cluster) to be identified as a GI.
- The presence of certain genes (e.g., integrases, transposases, phage genes) is associated with GIs [20].
- In addition to the size of the cluster, evidence from mycobacterial phages [39] suggests that the size of the genes themselves is shorter in GIs than in the rest of the bacterial genome. Different theories suggest that this may confer mobility or packaging or replication advantages [12].

- Some GIs integrate specifically into genomic sites such as tRNA genes, introducing flanking direct repeats. Thus, the presence of such sites and repeats may be used as evidence for the presence of GIs [40, 41, 42].

Other research suggests that the directionality of the transcriptional strand and the protein length are key features in GI prediction [12]. The available tools focus on one or more of the mentioned features.

PATRIC provides a compare region viewer service, that aligns a query gene against other related genes, and presents the pileups along with their neighborhoods graphically, allowing users to visualize the genomic areas of interest. An example output generated by this service is shown in Figure 2.6.


Feature View
Bacteria » Proteobacteria » Gammaproteobacteria » Enterobacterales » Enterobacteriaceae » Klebsiella » Klebsiella aerogenes » Enterobacter aerogenes KCTC 2190
fig|1028307.3.peg.135 | EAE_00715 | Cytoskeleton protein RodZ

Region Size: 10,000bp Number of genomes: 10 Method: PATRIC cross-genus families (PGfams) Genomes: Reference & Representative

Overview
Genome Browser
Compare Region Viewer
Transcriptomics
Correlated Genes
Interactions
Update
Export
(0) Enterobacter aerogenes KCTC 2190
(1) Klebsiella oxytoca strain CAV1374
(2) Klebsiella oxytoca KCTC 1686
(3) Klebsiella variicola At-22
(4) Klebsiella pneumoniae subsp. pneumoniae HS11286
(5) Klebsiella pneumoniae subsp. pneumoniae MGH 78578
(6) Klebsiella pneumoniae subsp. pneumoniae DSM 30104
(7) Raoultella ornithinolytica B6

Figure 2.6: Snapshot of the Compare Region Viewer service provided by PATRIC. The image shows a genomic region of the query genome (first row) aligned against a set of other genomes, anchored at the focus gene (represented as a red arrow). The service starts with finding other genes that are of the same family as the focus gene, and then aligns their flanking regions accordingly.

To ensure efficiency and consistency, we implemented an offline version of the visualization part. To generate the images, first the Compare Region service is called via PATRIC's command line interface. The call accepts parameters such as the query gene, region size, the set of genomes to be used for alignment, and the number of genomes to be displayed. We chose a region size of 10,000 base pairs, to be aligned against 20 genomes, using the set of representative and reference genomes found on PATRIC. The call to the Compare Region service returns information that includes the location, family, direction, and size of every gene in the region. This information is then transformed into the position, color, and size of the arrow representing each gene in the image. In a nutshell, Each gene is represented as an arrow, scaled to capture its size and strand directionality. Colors represent functionality. The red arrow is reserved for the query gene, which is placed in the middle of the first row, and at which the alignment with the rest of the genomes is anchored. Some colors are reserved for key genes: green for mobility genes, yellow for tRNA genes, and blue for phage related genes. By using these color-coded arrows of various sizes, the images capture the protein length, functionality, strand directionality, and the sporadic distribution of islands.

In the produced images, genomic islands appear as gaps in alignment as opposed to conserved regions. Figure 2.7 shows sample visualizations of different genomic fragments belonging to the two classes. Figures 2.7 (a) and 2.7 (b) are examples of a query genome with a non-conserved neighborhood. The focus gene lacks alignments in general or is aligned against genes with different neighborhoods than the query genome. In contrast, Figures 2.7 (c) and 2.7 (d) show more conserved regions, which are what we expect to see in the absence of GIs (labelled as continents in the image).

2.3.3 Transfer Learning

This kind of visual representation makes it easier to leverage the powerful machine learning technologies that have become the state of art in solving computer vision problems. Deep learning is the process of training neural networks with many hidden layers. The depth of these networks allows them to learn more complex patterns and higher-order relationships, at the cost of being more computationally expensive and requiring more data to work effectively. So, while PATRIC provides a lot of genomic data, the challenge comes down to building a meaningful training dataset. The databases available are still limited in size and specific in

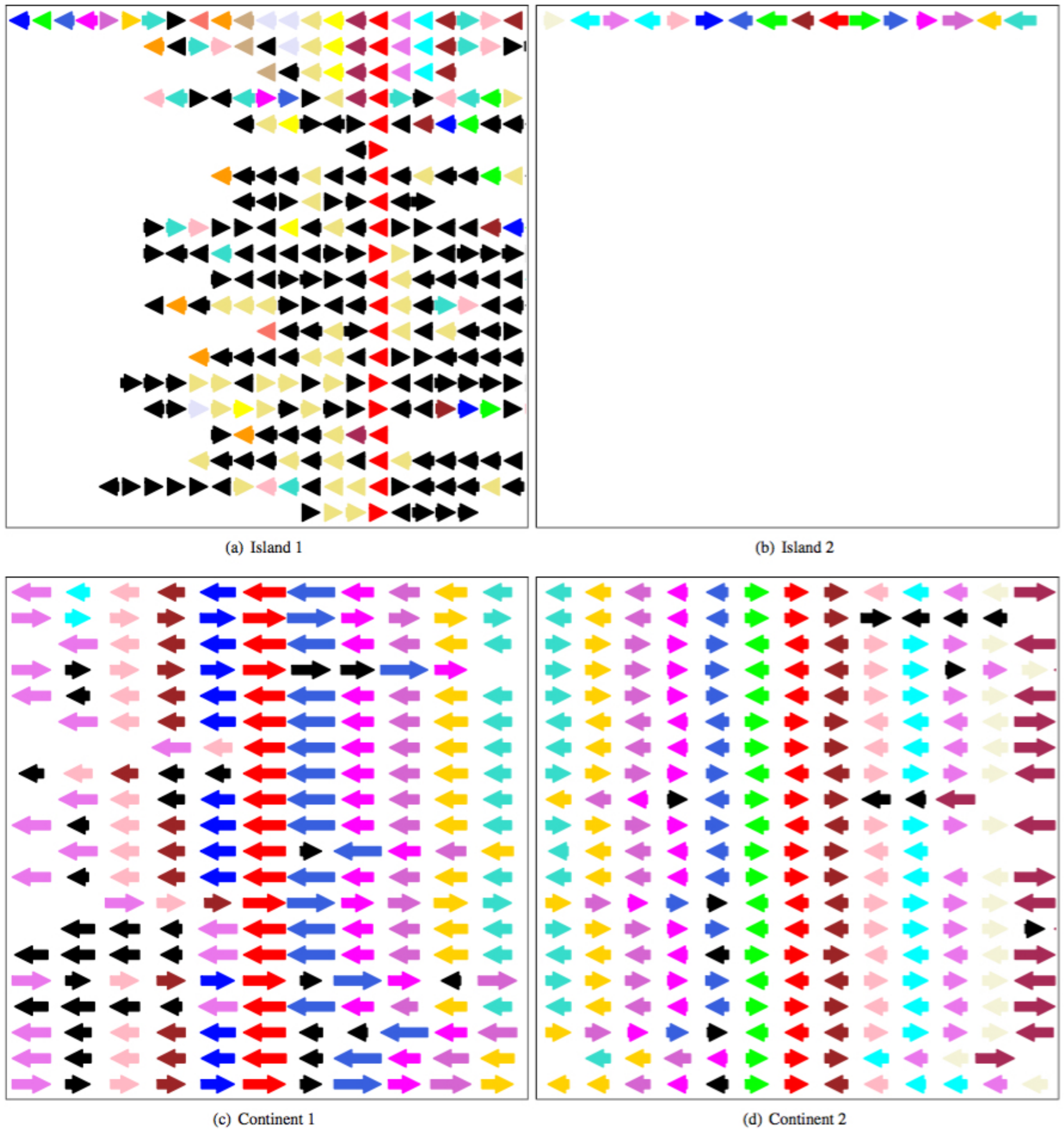


Figure 2.7: Examples of images generated using the compare region viewer. Each arrow represents a gene color coded to match its functionality. The first row is the genome neighborhood of the focus gene (red), and the subsequent rows represent anchored regions from similar genomes sorted by their phylogenetic distances to the query genome. Genes that would be labeled as not part of a genomic islands are referred to as continents in this figure.

content, which in turn limits the ability even for advanced and deep models to learn and generalize well. To avoid over-fitting, we applied transfer learning [43], by using Google's Inception V3 neural network architecture that has been previously trained on ImageNet [44]. Inception V3 is a 48-layer-deep convolutional neural network. Training such a deep network on a limited dataset such as the one available for GIs is unlikely to produce good results. The idea behind transfer learning is that a model trained on ImageNet is better than an untrained model initialized with random weights at visual recognition and feature extraction. By removing the top layer of the pre-trained model and training a new one on the GI dataset, the model can apply the low-level visual feature recognition learned using the much more extensive dataset towards the new task.

2.4 Discussion

We presented a new method, called Shutter Island, which demonstrates the effectiveness of training a convolutional neural network on visual representations of genomic fragments to identify genomic islands. In addition to using powerful technologies, our approach may add an extra advantage over whole-genome alignment methods because performing the alignment over each gene may provide a higher local resolution and aid in resisting evolutionary effects such as recombination and others that may have happened after the integration and that usually affect GI detection efforts.

One challenge in assessing GI prediction is getting precise endpoints for predicted islands. Since different tools report a different number of islands owing to the nature of the features they use, where one island could be reported as many or vice versa, we considered a tool to predict another's islands if any of its predictions overlap with the other tool's predictions. We counted the percentage of base pair coverage of that other tool as represented by its predicted endpoints. This allowed us to compare overlapping islands predicted by different tools even if their coordinates did not match.

Note that when assessing the tools' predictions, our definition of a true positive was different from the definition proposed in the Phispy study: where they define a true positive as a region with at least six phage related genes, we realize that the available datasets are much larger than what was accessible at the time of their publication, and thus the number six that was claimed to have been determined empirically may not be relevant anymore. We argue that if a region is suspected to be a GI, the mere presence of a phage related gene in that region adds to the confidence in its prediction as part of a GI. Moreover, we find that the database used by the study to determine phage functionality may be outdated, and we thus resort to certain keywords in the gene annotations generated by our system PATRIC to determine functionality. This is similar to Phaster's validation method, whereby the presence of certain keywords (e.g, caspid) is used to verify predictions made by the tool. To identify these keywords, we scoured the literature and identified certain gene annotations that are related to GIs. Such annotations of gene identity are either directly curated by humans or reflect human assessment through exemplar-based computational propagation. We constructed a standard vocabulary of the GI-related keywords that were also in agreement with the more extensive list of keywords used by Phaster for the same purpose.

Our initial inspiration for representing genome features as images came from observing how human annotators work. These experts often examine the "compare region" view for a long time before they decide on the gene identity. A critical piece of information they rely on is how the focus gene compares with its homologs in related genomes. This information is cumbersome to represent in tabular data because (1) explicit all-to-all comparison is computationally expensive; (2) the comparisons need to be done at both individual gene and cluster levels including coordinates, length, and neighborhood similarities; and (3) human experts integrate all these different levels of information with an intuition for fuzzy comparison, something that is hard to replicate in tabular learning without additional parameterization or augmentation. Representing genomic features as images mitigates all three issues.

Images offer a natural way to compare genes (horizontally) and clusters across genomes (vertically) with 2D convolution. The fact that the compare region view sorts genomes by evolutionary distance allows the neural network to exploit locality and place more emphasis on close genomes via incremental pooling. An additional benefit of working with images is to be able to leverage the state-of-the-art deep learning models, many of which were first developed in vision tasks and perfected over years of iterations. Google researchers have used spectrograms (instead of text) in direct speech translation [45] and DNA sequence pile-up graphs (instead of alignment data) in genetic variant calling [46]. In both cases, the image-based models outperformed their respective previous state-of-the-art method based on traditional domain features. Further, the low-level visual feature patterns learned in pre-trained image models have been demonstrated to transfer to distant learning tasks on non-image data in several preliminary studies ranging from environmental sound classification to cancer gene expression typing [47]. In the next chapter, we present another application that makes use of representing genomic data as images.

CHAPTER 3
DETECTING OPERONS (GENE CLUSTERS) IN BACTERIAL GENOMES VIA VISUAL REPRESENTATION LEARNING

In this chapter we demonstrate another application of direct visual representation of genomic data to predict operons in bacterial genomes. Contiguous genes in prokaryotes are often arranged into operons. Detecting operons plays a critical role in inferring gene functionality and regulatory networks. Human experts annotate operons by visually inspecting gene neighborhoods across pileups of related genomes. These visual representations capture the inter-genic distance, strand direction, gene size, functional relatedness, and gene neighborhood conservation, which are the most prominent operon features mentioned in the literature. By studying these features, an expert can then decide whether a genomic region is part of an operon. We use similar visual representations of genomic fragments to make operon predictions. Using transfer learning and data augmentation techniques facilitates leveraging the powerful neural networks trained on image datasets by re-training them on a more limited dataset of extensively validated operons. Our method (Operon Hunter) [48] outperforms the previously reported state-of-the-art tools, especially when it comes to predicting full operons and their boundaries accurately. An additional advantage of this approach is that it makes it possible to visually identify the features influencing the network's decisions to be subsequently cross-checked by human experts.

3.1 Introduction

3.1.1 Background

Genes in prokaryotic genomes assemble in clusters, forming transcription units called operons (Figure 3.1 [49]). These genes share a common promoter and terminator [50], and are usually metabolically or functionally related. Predicting operons helps understand high level

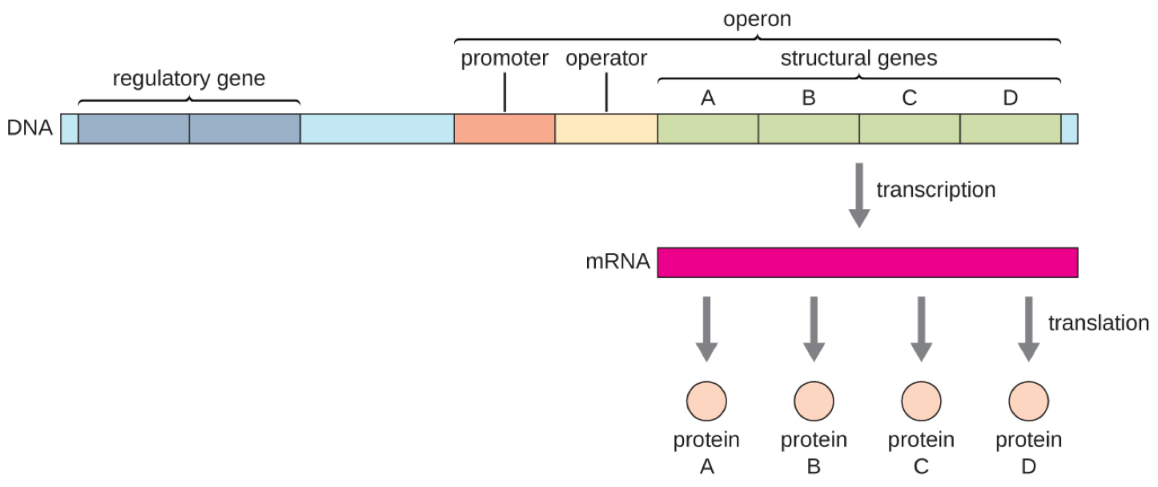


Figure 3.1: A visualization of an operon in a bacterial genome. An operon is a group of genes that share the same promoter and terminator, and are thus transcribed together. Adapted from (Nina Parker et. al. 2016 [49]).

organization of genes and regulatory networks [51, 52, 53, 54, 55, 56, 57], annotate gene functions [58], develop drug candidates [59], and inhibit antibiotic resistance [60]. While operons are prevalent in bacterial genomes, their detection is challenged by a multitude of factors that contribute to their organization.

Human experts annotate operons by visually inspecting stretches of genes in a comparative genomics browser (see an example in Figure 2.6). Such visual representations enable synthesis of two sources of information. First, gene-level features such as size, strand direction, function label, and inter-genic distance can be checked for consistency by scanning contiguous regions within the same genome. Second, close relatives of the query genome can be retrieved, aligned, and anchored against the focus gene, which allows human experts to see whether there is evolutionary evidence of regional conservation. This second dimension of criteria is critical to the operon call, but is often difficult to quantify, requiring human judgment on selecting phylogenetic distance and weighing the similarity of genomic neighborhoods.

Several tools have been proposed to detect operons computationally, yet they mostly focus on gene-level features. Even when phylogenetic information is added, it's preprocessed and presented in a tabular form, preventing flexible learning. We are inspired by how human experts work and the recent advance in deep learning to explore a novel approach based on visual learning. We hypothesize that, when presented with all evidence in a similarly visual representation of genomic images that humans rely on, neural networks can learn to capture the complex operon determinants within and across genomes. To this end, we present a method, named Operon Hunter, that predicts operons from visual representations of genomic fragments. Our method uses a pre-trained network via transfer learning to leverage the power of deep neural networks trained on image datasets. The network is re-trained on a limited dataset of extensively validated and experimentally verified operons. We compare our method with the state-of-the-art operon predictors. Our results show that Operon Hunter outperforms them in identifying full operons as well as delineating operon boundaries. Furthermore, our visual approach generates insights into regions of importance that can be cross-checked by human experts.

3.1.2 Related Work

Different methods focus on different operon features to make their predictions. Some methods use Hidden Markov Models (HMMs) to find shared promoters and terminators [61, 62, 63]. Other methods rely on gene conservation information [64], while others leverage functional relatedness between the genes [51, 65]. The most prominent features used in operon prediction are transcription direction and inter-genic distances, as reported in the literature [51, 53, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76]. Gene conservation is another important feature, since adjacent genes that are co-transcribed are likely to be conserved across multiple genomes [55, 77]. Different machine learning methods are used to predict operons, such as neural networks [67, 68], support vector machines [70], and decision tree-based classifiers [69]. Other tools utilize Bayesian probabilities [71, 72, 73], genetic algorithms [74], and graph-theoretic techniques [65, 72].

We focus our attention on two machine learning based tools. The first tool [78] developed by Zaidi and Zhang is reported to have the highest accuracy among operon prediction methods. It is based on an artificial neural network that uses inter-genic distance and protein functional relationships [53]. To infer functional relatedness, this method uses the scores reported in the STRING Database [79]. The STRING database captures functional relatedness through scores generated using information about gene neighborhood, fusion, co-occurrence, co-expression, protein-protein interactions, in addition to information extracted by automatic literature mining. The predictions made by this method were compiled into what is called the Prokaryotic Operon Database (ProOpDB) [54], and released later as a web service called Operon Mapper [80]. For simplicity, we will refer to this method as ProOpDB, given that it was our resource for the predictions over the model organisms we used for the cross-tool comparison. The second tool is called the Database of Prokaryotic Operons (Door) [52] and was ranked as the second best operon predictor after ProOpDB by the same study [78]. It was also ranked as the best operon predictor among 14 tools by another independent study [81]. Door's algorithm uses a combination of a non-linear (decision-tree based) classifier and a linear (logistic function-based classifier) depending on the number of experimentally validated operons available for the genomes used in the training process. Among the features Door uses to perform its predictions for adjacent gene pairs are: the distance between the two genes, the presence of a specific DNA motif in the genomic region separating them, the ratio of the genes' lengths, the genes' functional similarity (determined using Gene Ontology (GO)), and the level of conservation of the genes' neighborhood.

3.2 Results

Most operon prediction tools make their predictions on isolate gene-pairs. Contiguous gene pairs predicted to be as part of an operon are then aggregated to form full operon predictions [51]. We refer to gene pairs that are part of an operon as operonic, and gene pairs where one is a boundary of an operon, or that include a verified operon consisting of a single gene, to be non-operonic. Our validation dataset consists of two well-studied genomes with experimentally validated operons: E. coli and B. subtilis. These genomes are extensively used to verify the majority of the published tools' results. We compare the performance of the tools first when considering gene-pair predictions, and then considering full-operon predictions. While Operon Hunter demonstrates an advantage in both, the advantage is more pronounced when assessing the ability to predict full operons and their boundaries accurately.

3.2.1 Evaluation of Gene-Pair Predictions

We report the sensitivity (true positive rate), precision, and specificity (true negative rate) for the models' performance over gene pair predictions in Table 3.1. These statistical measures were calculated according to the following definitions:

$$\text{Sensitivity} = \frac{TP}{TP + FN} \tag{3.1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.2}$$

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{3.3}$$

Where TP is the number of operonic pairs predicted correctly (true positives), FP is the number of non-operonic pairs incorrectly predicted as operonic (false positives), FN is the number of operonic pairs incorrectly predicted as non-operonic (false negatives), and TN is the number of non-operonic pairs predicted correctly (true negatives).

Table 3.1: Sensitivity, precision, and specificity. Sensitivity (true positive rate) is the percentage of operonic gene pairs that were detected by the different tools, precision is the percentage of operonic gene pairs predicted by the different tools which are actually true positives, and specificity (true negative rate) is the percentage of non-operonic gene pairs that were detected by the different tools. For sensitivity and specificity, results are first shown per genome, and then as an aggregate over the entire testing dataset.

	Sensitivity E. coli (361 pairs)	Sensitivity B. subtilis (369 pairs)	Sensitivity Aggregate (730 pairs)	Precision Aggregate (730 pairs)	Specificity E. coli (461 pairs)	Specificity B. subtilis (302 pairs)	Specificity Aggregate (763 pairs)
Operon Hunter	88%	**97%**	92%	92%	**95%**	88%	92%
ProOpDB	**93%**	93%	**93%**	89%	90%	88%	89%
Door	81%	86%	84%	**94%**	94%	**97%**	**95%**

ProOpDB scores the highest sensitivity but the lowest precision. The opposite is true for Door, which achieves the lowest sensitivity but the highest precision. Operon Hunter's performance is more stable across the two metrics. To capture the predictive power of the model on both classes in a single metric, we report the F1 score, accuracy, and the Mathews Correlation Coefficient (MCC) in Table 3.2, calculated using the following definitions:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.4}$$

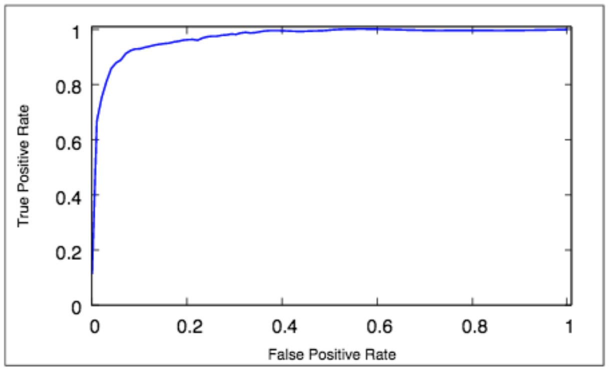
$$\text{MCC} = \frac{TP\text{x}TN - FP\text{x}FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.5}$$

$$\text{F1} = 2 \text{ x } \frac{Sensitivity \text{ x } Precision}{Sensitivity + Precision} \tag{3.6}$$

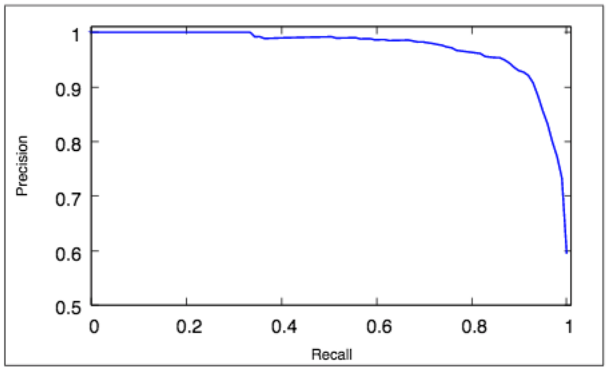
Table 3.2: Accuracy, MCC (Mathews Correlation Coefficient), and F1-score achieved by the different tools over the entire testing dataset.

Tool	Operon Hunter	ProOpDB	Door
Accuracy	**0.92**	0.91	0.89
MCC	**0.84**	0.82	0.79
F1-score	**0.921**	0.908	0.886

Operon Hunter scores the highest on all metrics measured, followed by ProOpDB then Door. We also show the Receiver Operating Characteristic (ROC) curve and the Precision-Recall curve with the corresponding Area Under the Curve (AUC) in Figure 3.2. Evaluated on the entire testing dataset, Operon Hunter scores a ROC AUC and a Precision-Recall AUC of 0.97.



(a) Receiving Operating Characteristic (ROC) curve



(b) Precision-Recall curve

Figure 3.2: The Receiving Operating Characteristic (ROC) curve (a) and Precision-Recall curve (b) corresponding to Operon Hunter evaluated over the entire testing dataset. The Area Under the Curve (AUC) for both is 0.97.

3.2.2 Evaluation of Full Operon Predictions

Predicting full operons is a more challenging task that requires the accurate identification of the operon endpoints. We reserve the definition of operons as clusters including at least two genes. Matching earlier reports concerning different methods [78], the accuracies of all three tools drop when validated on full operon predictions rather than separate predictions made for every gene pair. We present the percentage of operons predicted fully by each of the tools in Table 3.3. Full matches are operons reported in the literature that are accurately predicted with the same gene boundaries as the published endpoints. We only consider operons that consist of more than one gene, and were reported in the Operon Database (ODB) in addition to RegulonDB [82] or DBTBS [83]. RegulonDB is a database containing verified operons found in E. coli, and DBTBS is a database that contains verified operons found in B. subtilis. This amounted to 254 full operons. Predictions that only partially match a verified operon are not shown. To make full operon predictions, the model starts by generating a prediction for every consecutive gene pair in a genome. In a second pass, pairs predicted as operonic are then merged into full operons. Operon Hunter accurately identifies 85% of the operons fully, which is the most across the three tools. Both ProOpDB and Door show a drop in accuracy, to 62% and 61% respectively.

Table 3.3: Comparison of the results between OperonHunter, ProOpDB, and Door when considering full operon predictions. Exact Operon Matches are the percentage of operons predicted where the endpoints exactly match those of the experimentally verified operons. The percentages are reported over 254 full operons that consist of more than one gene and are reported in RegulonDB/DBTBS and ODB.

	Operon Hunter	ProOpDB	Door
Exact Operon Matches	**85%**	62%	61%

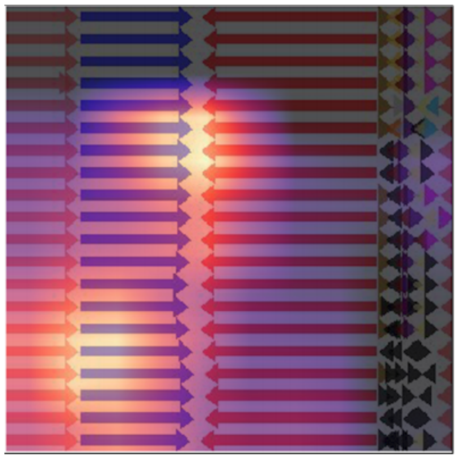
3.2.3 Cross Validating Visual and Operon Features

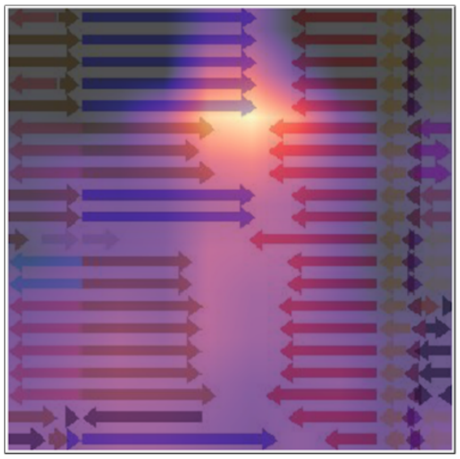
An advantage of visual models is that they can generate insights into decision making by highlighting regions of importance. Such interpretable representations reveal inner workings of the model and can be checked by experts to see if they ground their intuition. To investigate the model's performance, we used the Grad-Cam [84] method to overlay heat-maps over the input images, highlighting the areas of attention that most affect the network's decisions. Figure 3.3 shows some of the network's confident predictions.

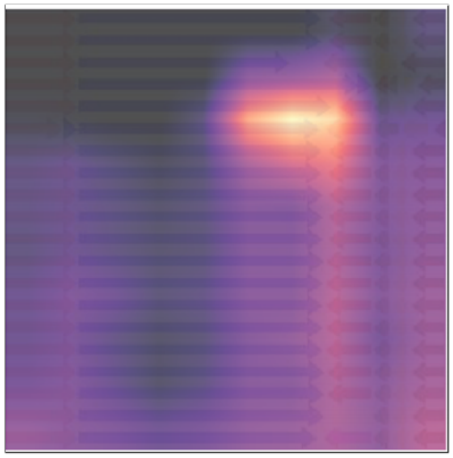
It appears that when predicting a gene pair as non-operonic, the model bases its decision on the entirety of the input image, whereas it focuses on the immediate vicinity of the gene pairs predicted as operonic. Upon closer inspection, we speculate that the following are the most important features influencing the model's decision:

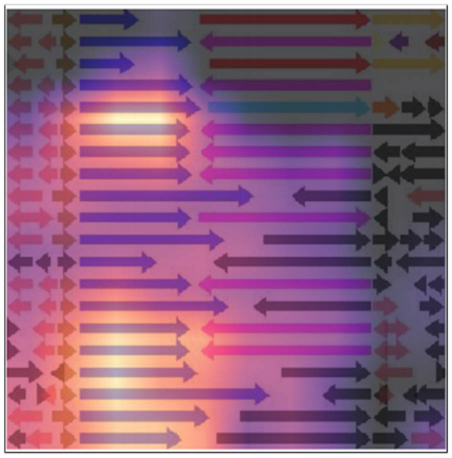
- Figure 3.3 (a): The strand directionality of the gene pair.
- Figure 3.3 (b): The mis-alignment between the genomes.
- Figure 3.3 (c): The low alpha channel of the image, which is a representation of the low STRING score between the query gene pair.
- Figure 3.3 (d): The non-conservation of the query gene pair neighborhood.

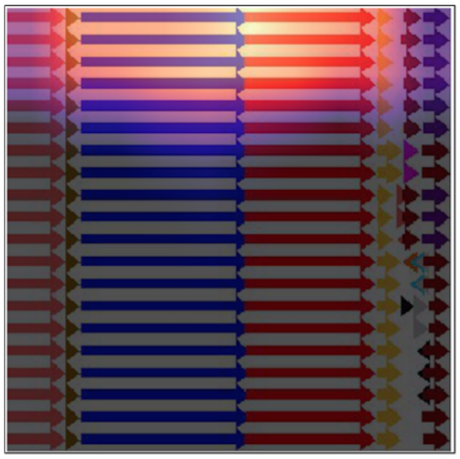
All these features were previously mentioned as among the most prominent in operon identification. Operonic gene pairs show more overall conformity across the input image, with flanking regions being mostly aligned/conserved, and query gene pairs having little or no inter-genic distance, and usually similar directions, as shown in Figure 3.3 (e). Even when


(a) Non-Operon: Opposite strand direction


(b) Non-Operon: Non-conserved alignment


(c) Non-Operon: Low STRING score


(d) Non-Operon: Non-conserved gene neighborhood


(e) Operon: Intergenic distance, strand direction

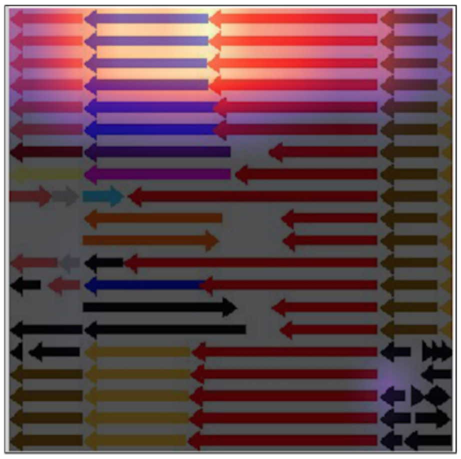

(f) Operon: Intergenic distance, strand direction

Figure 3.3: HeatMaps generated using the Grad-Cam method and overlayed over input images, highlighting the network's areas of attention that had most influence over the network's decision. Each sub-figure shows the network's correctly predicted label, with what we believe to be the most prominent feature leading to the network's decision.

the neighborhood of the query gene pair is not conserved, as shown in Figure 3.3 (f), the network seems to base its decision on the similar strand direction and lack of an inter-genic distance between the gene pair. Thus, it seems that when predicting gene pairs as operonic, the network focuses on the immediate vicinity of the pair, disregarding other areas of the image. However, the network seems to pick up on anomalies around the gene pair, taking the entirety of the image into consideration when predicting non-operonic images.

3.3 Methods

3.3.1 Datasets

We used the "Known Operons" section of the Operon DataBase (ODB) [85] to construct the training dataset. The Known Operons section contains a list of operons that are experimentally verified, which we used to label the generated images. Out of the entire dataset, the six genomes (Table 3.4) with the most number of labeled operons were selected. These genomes have significantly more operons than the other genomes listed in this section of the database. For each of the selected genomes, our program produces an image for every consecutive pair of genes that are part of a validated operon. To generate the images, each of these genomes is aligned with the set of reference+representative genomes found on PATRIC. For every aligned gene, an image is generated to capture the surrounding 5 kilo base pairs (Kbp) flanking region. The resulting dataset consisted of 4,306 images of operonic gene pairs. To generate the dataset representing the non-operonic gene pairs, we used the standard approach reported by ProOpDB [54] as follows: Genes that are at the boundaries of known operons are labeleled along with the respective upstream or downstream gene that is not part of that operon as a non-operonic gene pair. We skipped single-gene operons from the training dataset. A balanced dataset was then curated from the total set of images created.

Table 3.4: Breakdown of the training dataset: The genome names and the corresponding number of gene pairs used. Operon pairs were harvested from the Known Operons section of the Operon Database (ODB).

Genome name	Operon pairs	non-Operon pairs
Escherichia coli	1,443	1,322
Listeria monocytogenes	806	780
Legionella pneumophila	611	791
Corynebacterium glutamicum	525	396
Photobacterium profundum	447	544
Bacillus subtilis	474	457
Total	4,306	4,290

3.3.2 Feature Encoding via Visual Representation

Similar to the approach presented in Chapter 2, we tweaked our offline implementation of the Compare Region Viewer service offered by PATRIC to generate the images (an example of which is shown in Figure 3.4).

While the general idea is similar, there are some notable differences in the generated images. Each row of arrows in the generated image represents a region in a genome, with the query genome being the top row. Each arrow represents a single gene, scaled to reflect its size relative to its region, in addition to the gene's strand directionality. The distances between the genes are also scaled on each row relative to the gene's region. Each image consists of three regions (although they are not explicitly divided). The central region makes up two thirds of the image, and represents the query gene pair. Genes that fall before the query gene pair are represented to the left of the central region, and those that fall after the query gene pair are represented to the right of the central region. Dividing the image implicitly into three regions highlights the area of most interest by zooming in on the query gene pair, while preserving the relevant conservation information of the flanking genomic fragments on the sides. Colors represent gene functionality. The blue and red arrows are reserved to represent the query gene pair and the rest of the genes that belong to the same families. In general, genes share the same color if they belong to the same family, or are colored black after a certain color distribution threshold. The families used in the coloring process are PATRIC's

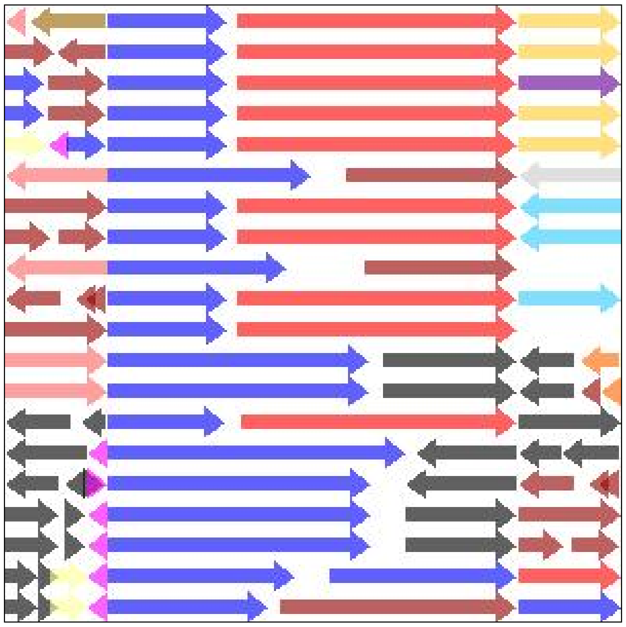


Figure 3.4: Example of an image generated by our offline version of the Compare Region Viewer service to be fed as input to the neural network. Each arrow represents a single gene. Each row captures the area of interest in a genome. The query genome is the top row. The rest of the rows are genomes selected by evolutionary distance. The query gene pair are colored blue and red. Genes share the same color if they belong to the same family and that family. The query gene pair are centered in the middle, occupying 2/3 of the image's width. The rest of the flanking region is represented correspondingly to the left/right of the center region. The alpha channel of the image is the STRING score of the query gene pair.

Global Pattyfams that are generated by mapping signature k-mers to protein functionality, using non-redundant protein databases built per genus before being transferred across genera [86]. Finally, the image's alpha channel is set to be equal to the STRING score of the query gene pair. If no score exists, the default alpha channel is set to a minimum of 0.1. The generated images capture most of the prominent features mentioned earlier, such as gene conservation, functionality, strand direction, size, and inter-genic distance.

3.3.3 Transfer Learning

The relatively small size of the dataset combined with the depth of the network pose the risk of over-fitting. We resort to transfer learning and data augmentation to avoid that. To train and test our model, we used the FastAI [87] platform. The best performance was observed using the ResNet18 model previously trained on Imagenet. As mentioned earlier, this is more powerful than starting with a deep network with random weights. Another technique that is commonly used against over-fitting is data augmentation, whereby the training dataset is enriched with new examples by applying certain transforms on the existing images. These transforms include flips (horizontal and vertical), zooming effects, warping, rotation, and lighting changes. Using FastAI's toolkit, we augmented our training dataset by allowing only one transform, the horizontal flip, to be applied. Given the deliberate feature engineering processes taken to create the input images, we believe that horizontal flips could safely be applied without altering the true nature of the images. This would keep the key information intact and in place (e.g. by keeping the query genome as the top row).

3.3.4 Model Validation

We resort to two extensively studied genomes with experimentally verified operons: E. coli and B. subtilis. These genomes are the standard for verifying operon prediction results in the literature. We limit our validation by using only the experimentally validated operons found in these genomes. We compare the predictions made by Operon Hunter to those made by ProOpDB and Door, the tools with state of the art accuracies as reported by independent studies [52, 54, 78]. To build the testing dataset (Table 3.5), we used the operons published in RegulonDB and DBTBS. We cross check the operons found in these databases, with the ones published in the Known Operons section of ODB. We selected the operons that are reported in both: The database corresponding to its organism (RegulonDB for E. coli, DBTBS for B. subtilis), and ODB. The resulting dataset consists of 730 operonic gene pairs. We used the same approach mentioned earlier to construct a non-operonic dataset. Namely, the boundaries of the operons were selected along with their neighboring genes as non-operonic pairs. To construct the non-operonic datasets, it was enough for an operon to be published in any of the mentioned databases to be considered, resulting in a slightly larger non-operon dataset. Using operons that were experimentally verified and published in multiple independent databases adds confidence to the assigned label.

Some tools train on the same organisms they report their predictions on. In our experiments, we exclude the testing dataset from the training dataset, by following a leave-one-out approach on the genome level. Thus, none of the genomes used in the training process belong to the same organism as the genome used for testing our method's performance. For example, when evaluating the model's predictions on the E. coli dataset, it is trained on all the images representing all the genomes except E. coli, and then tested on the images representing the E. coli genome. This approach leads to an 87-13 train-test split for the E. coli genome, and a 92-8 train-test split for the B. subtilis genome.

Table 3.5: Breakdown of the testing dataset: The genome names and the corresponding number of gene pairs used. Operon pairs were scoured from RegulonDB (for E. coli) and DBTBS (for B. subtilis) and matched with the Known Operons section of the Operon Database (ODB).

Genome	Operon pairs	non-Operon pairs
Escherichia coli str. K-12 substr. MG1655	361	461
Bacillus subtilis subsp. subtilis str. 168	369	302
Total	730	763

3.4 Discussion

We have presented a novel approach to operon prediction by training a deep learning model on images of comparative genomic regions. This approach has achieved better results than the currently available state-of-the-art operon prediction tools. In particular, it has demonstrated a clear advantage in predicting accurate operon boundaries (i.e. predicting operons fully).

An advantage of our model is in its effective synthesis of gene-level and phylogeny-level evidences. Traditional models come with preprocessed features and may be rigid in picking parameters on neighborhood size, genomic distance, or similarity metrics for comparing regions across genomes. The PATRIC compare region viewer is perfected by human experts, and strikes a balance in diversity and granularity in the way it brings representative genome relatives to an annotator's attention. Critically, these images give machine learning models a two-dimensional view of all relevant information without limiting them to a pre-determined way of encoding features. High-capacity neural network models are thus allowed a more flexible space to learn and pick up interactive features.

We experienced technical difficulties when trying to retrieve the operon predictions made by both tools we are comparing our performance against. Since the tools generate predictions for gene pairs, the shortest possible operon should thus consist of at least two genes. However, this conflicts with the reported predictions made by these tools, as they include operons consisting of only a single gene. We considered single-gene predictions to be part of a non-operonic gene pair. The tools report predictions for only a subset of the genes in a given genome, which leaves many genes without an assigned label. We decided to label the genes with missing predictions as non-operonic, since leaving them out of the performance measures would drastically diminish the tools' scores. This way, the metrics involving true negatives reported for the tools are the highest they could attain, assuming they would predict all the non-operonic gene pairs correctly. In a good faith attempt to replicate the work done by ProOpDB, we fed a neural network with the same architecture mentioned in their paper the same features (i.e, the inter-genic distance and STRING score for consecutive pairs of genes). We used our training dataset, which we believe to be richer, considering that they were training on only one organism (E. coli or B. subtilis). We fine tuned different hyper parameters, like the learning rate and the number of epochs, and experimented with different activation functions before settling on the one mentioned in their paper. Instead of generating STRING-like scores for the genes that miss that feature, we followed our previous approach of using a default value of 0.1. It is worth mentioning that the STRING database has been updated since the time of their first publication, and the current version has score assignments for most of the operonic gene pairs ($> 99\%$ of the pairs used in our training and testing datasets). Following this approach, we achieved a slightly lower true positive rate, with a slightly higher precision, leading to the same F1 score previously reported for their tool, and a slightly higher specificity, that is still the lowest across the three tools we used in our comparison.

We point out some of the challenges that undermine operon prediction in general. One limitation that faces predictors that rely on features such as gene conservation or functional assignment is the requirement to have such information about all the genes in a genome. So while such predictors might perform well on gene pairs that include the necessary features, their performance might drop considerably when making predictions over the entire genome [51]. Moreover, even though most methods validate their results by comparing their predictions over experimentally verified operons, the fact that the experimentally verified datasets are only available for a small subset of the sequenced genomes and that the datasets used vary between studies poses extra challenges making the comparison between the available tools non-straightforward. Brouwer et. al. tried to compare several methods using a uniform dataset and noticed a significant gap between the measures achieved and those reported in their original papers. The drop in performance was even higher when considering full operon predictions rather than separate gene pair predictions [78, 81]. Finally, Some methods include the testing dataset as part of the training dataset, which leads to a reported accuracy that is significantly higher than what would be otherwise, given that the flow of information taking place in the training process would not be easily and readily transferable to novel genomes used as a testing dataset [53]. Some methods report a decrease in accuracy when the training and testing datasets belong to different organisms. In fact, the accuracy reported by many of the available tools drop anywhere between 11 and 30% when the training data and the operon predictions correspond to different organisms [53, 81]. This lack of generalization places severe limitations on the methods' applicability, especially when treating novel genomes. Even for known genomes, this poses the additional challenge of requiring more ground-truth data for operons in other genomes that belong to the same organism. For example, as mentioned earlier, Door switches between a linear model and a more complex decision-tree based model depending on the availability of the experimentally verified operons in the organism of the query genome that can be used for training the model. Our approach alleviates these challenges and generalizes successfully across organisms. As Table 3.4 shows, the genomes constituting our training dataset span different genera, but the performance of Operon Hunter when predicting operons in both E. coli and B. subtilis does not vary significantly. This is due to the fact that in the compare region viewer service that we use to construct the images, while the genomes aligned with the query genome are chosen based on evolutionary distance, they are eventually represented as images using a more generic method, that captures all underlying relevant operon features.

Much like feature engineering methods, casting tabular data to images encodes information in a way more amenable to learning without explicitly adding information. It can also be easily integrated with other data modalities in the latent vector representation to prevent information loss. We hypothesize this emerging trend of representing data with images will continue until model tuning and large-scale pre-training in scientific domains start to catch up with those in computer vision. The key principles that should guide the feature transformation are highlighting the entities and the relationships between them. Applications of this method are especially useful when the features are not easily quantifiable, as is the case in any application involving comparative genomics. Due to the generic nature of the visualizations and the available data augmentation techniques, we expect applications similar to our method to transform genomics problems where ground truth datasets are limited.

CHAPTER 4

TABULAR FEATURE TRANSFORMATION AND MODULAR LEARNING

In chapters 2 and 3, we demonstrated the effectiveness of transforming features by representing them in a way that highlights entities and the relationships. The transformations applied were in the form of visual representations engineered using our domain knowledge about the problem and the features it entails. In this chapter, we turn our attention to tabular feature transformations and propose a general approach with a pipeline to automate it. The transformed feature vector will be either a partitioning of the original vector, where every group of features represents an entity or a relationship, or a permutation of the original feature vector placing related features as neighbors. The simplest way to transform the features is by presenting the separate entities and relationships as separate partitions when domain knowledge that defines these entities is available. In the absence of domain-knowledge, one way to automatically group the features is by testing the predictive power of each subset of features in predicting the outcome, and using that to assign group scores and infer feature groups accordingly. Such approaches are intractable and infeasible as their computational complexity will be on the order of the power set of the set of features. We propose a greedy approach that uses the correlation values between the feature vectors to determine feature groups. In the cases where no feature groups emerge, our method generates a permutation of the input feature vector, where correlated features are placed as neighbors. The different transformations are matched with artificial neural network architectures that can leverage the corresponding properties in the data representation. When the transformation is a partitioning of the feature vector, we use a modular MLP (described later in this chapter) to accept the different feature groups through separate input layers. When the transformation is a permutation of the feature vector, we use a one-dimensional convolutional neural network to exploit the locality among the correlated features.

One advantage of performing the feature transformations within the tabular representation space is the computational efficiency compared to a transformation into the visual space (images require more memory, visual learners more compute power). An outline of our approach is illustrated in Figure 4.1.

4.0.1 Related Work

The visual transformations applied in chapters 2 and 3 are human engineered and designed to highlight the features relevant to the learning task. Other approaches seek to automate the feature transformation into the visual space by mapping tabular features into pixels. Their idea is to use statistical metrics to assess feature relationships, and place related features as neighboring pixels. The resulting visual representation can then be used as an input to a 2D CNN that is already adept at exploiting spatial locality, which now includes the neighboring pixels that are related. DeepInsight [88] is one such approach that uses non-linear dimensionality reduction techniques, namely as t-SNE [89] or kernel principal component analysis (kPCA), and applies the convex hull algorithm to convert tabular features into pixels where similar features are placed as neighboring pixels and dissimilar ones are placed further apart. REFINED [90] is another method that uses the euclidian distance between feature vectors, and a Bayesian Multidimensional Scaling (BMDS), and apply a hill climbing algorithm, to transform the tabular features into pixels. Their idea is to use 2D CNNs on the resulting images, which exploits spatial correlation and a reduced number of parameters compared to a fully connected network. The paper claims that 1D CNNs would not be effective in cases where the order of the features does not describe their dependencies. Image Generator for Tabular Data (IGTD) [91] is another method to transform tabular data into images by minimizing the difference between the ranking of distances between features and the ranking of distances between their assigned pixels to find an optimized pixel assignment. Han et. al. [92] convert tabular features into synthetic images to be used for classification tasks. They do that using the correlation values between features, and between features and

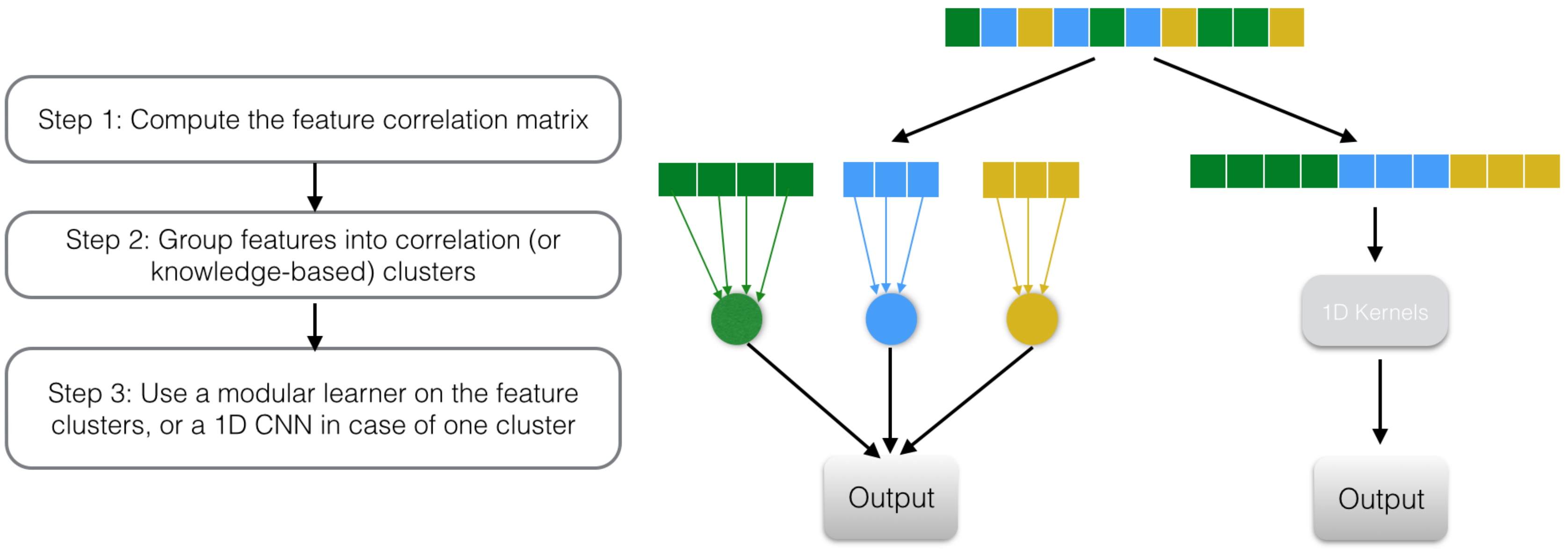

Step 1: Compute the feature correlation matrix
Step 2: Group features into correlation (or knowledge-based) clusters
Step 3: Use a modular learner on the feature clusters, or a 1D CNN in case of one cluster
Output
1D Kernels
Output

Figure 4.1: Schematic of our approach: (1) If the feature vector represents known entities and relationships that can be clustered using domain knowledge, the features are fed to a modular MLP in their clustered form. (2) If no domain-knowledge clusters are known, the feature vector is re-ordered based on the feature correlation matrix. (3) If correlation based clusters emerge (groups of clusters that show strong inter-group correlation values and weak outer-group correlation values), those features clusters are presented to a modular MLP. (4) If no correlation clusters emerge, the re-ordered feature vector is used as input to a one-dimensional convolutional neural network.

labels, and 0/1 optimization to maximize not only local but also global correlation in the new representation of the data. Torumoy Ghoshal [93] proposes a method to transform non-image data to have image-like properties and introduce the Hungarian Method to prevent information loss in the transformation process.
We argue that a transformation from tabular features to pixels in methods similar to the ones mentioned, are not only unnecessary, but add a computational burden that could be mitigated by using similar distance metrics to re-order features and exploit locality in a tabular setting by using a 1D CNN, in cases where domain knowledge is lacking and prevents a meaningful human-engineered grouping of related features. Thus, while our approach bears resemblance to the other approaches mentioned in this section, it remains a much simpler method, both conceptually and computationally.

Other approaches exist that do not perform feature transformation from a tabular to a visual domain. In a method called TABNN [94], the motivation is similar to our approach in that the proposed method aims to leverage feature grouping and a reduced model capacity to improve the learning performance, especially as compared to fully connected MLPs that might suffer from complex optimization hyper-planes and thus a higher risk of over-fitting. They attempt to automate the feature grouping using Gradiant Boosted Decision Trees (GBDT), and then reduce the feature groups to encourage parameter sharing and reduce the complexity of the resulting model. After clustering the feature groups, they use a recursive encoder with shared embedding to design the neural network architecture, and transfer structured knowledge from the GBDT to initialize the resulting neural network. Talip Ucar et. al. SubTab [95] introduces a framework that divides the input features into multiple subsets, and attempts to reconstruct the data from the subsets, arguing that this leads to a better learning of the latent representation than methods that use auto-encoders on corrupted versions of the data. Inspired by click-through-rate prediction problems, which deal with high dimensional and sparse datasets, Weiping Song et. al. AutoInt [96] is a method that automatically learns the high-order feature interactions of input features using a multi-head self-attentive neural network with residual connections. Golinko et. al. [97] utilize a feature embedding approach they developed, that is similar to principal component analysis in that the original features are correlated with the embedded features. Then they train a 1D CNN on the embedded data, and pop off the last layer to obtain a new representation of the features, which is then used as input to a classical machine learning algorithm (support vector machine, random forest, or nearest neighbor). Arlind Kadra et. al. [98] claim that well regularized MLPs significantly outperform state of the arts method devised to learn from tabular data, and use a combination of 13 regularization techniques in their method.

TabNet [99] is a method developed at Google, that uses sequential attention to select the most salient features per learning instance. Its architecture includes a feature transformer and an attentive transformer. By forming feature clusters, our method mimics feature selection approaches without discarding the features that are not chosen as strong predictors. As part of the empirical evaluations of our approach, we use some of the datasets mentioned by DeepInsight, REFINED, and TabNet in their corresponding papers, and compare our approach's performance to theirs in Chapter 6.

The rest of this chapter introduces our approach in more detail.

4.0.2 Transforming Tabular Features by Permuting them Using Feature Correlations

The first step in our approach is to compute the Pearson correlation matrix for all feature vectors in the training dataset. We then follow a greedy approach to create a permutation of the feature vector. We will refer to the original feature vector as $\mathbb{O}$ and the permuted feature vector as $\mathbb{P}$. Below are the steps followed to create the ordered set of features $\mathbb{P}$:

1. Start with $\mathbb{P}$ containing the first feature in $\mathbb{O}$.
2. Find the feature in $\mathbb{O}$ that is most correlated with the last feature added to $\mathbb{P}$.
3. Append that feature to $\mathbb{P}$ if it is not already there.

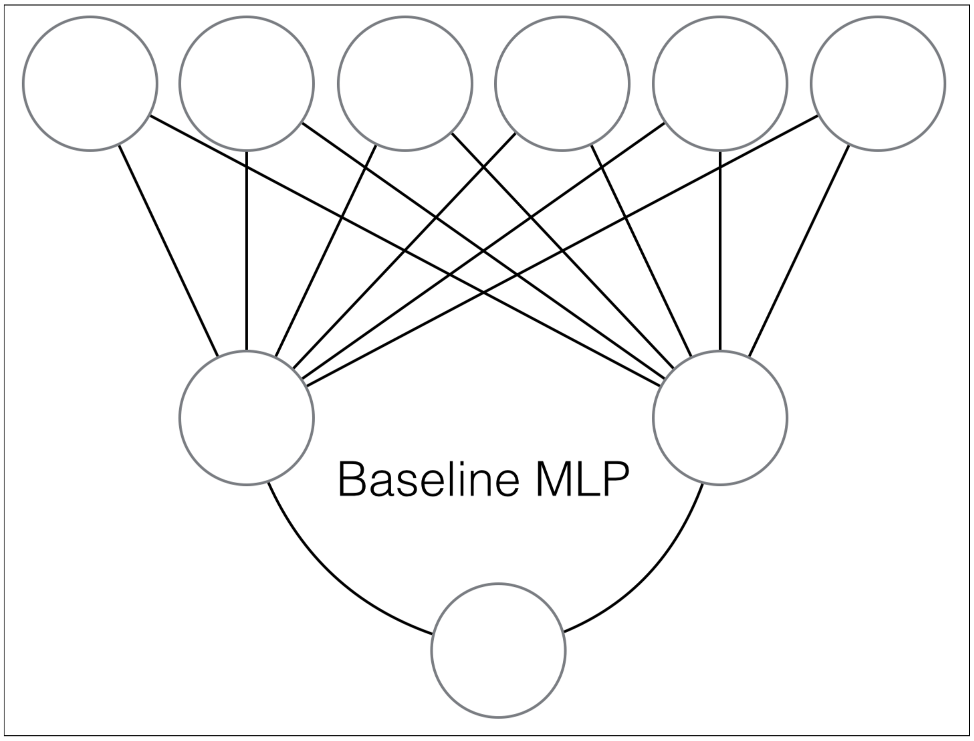
4. Repeat steps 2 and 3 until $\mathbb{P}$ contains all features in $\mathbb{O}$.

Note that our method does not distinguish between positive and negative correlation, but uses the magnitude of the correlation to determine the most correlated features.

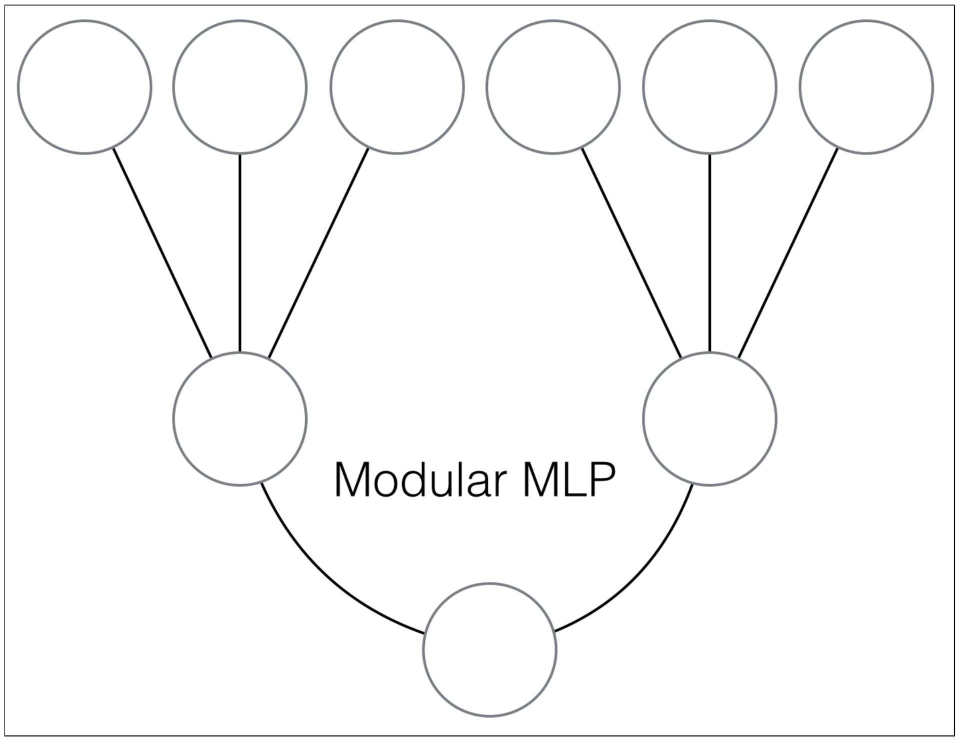
4.0.3 Transforming Tabular Features into Partitions Using Feature Correlations

The main idea behind our approach is presenting the input features as separate groups, each representing an underlying entity or relationship in the raw data. In the cases where we have domain knowledge that can be used to determine those feature clusters, we pass the feature groups to a modular MLP learner, that mainly differs from a baseline MLP by having multiple input layers each accepting a different feature group, and that concatenates the corresponding hidden layers before adding more connections. Figure 4.2 illustrates these two different MLP architectures.

With the lack of knowledge-based feature clusters, we attempt to form correlation-based feature clusters. To form the clusters, we use a greedy approach that introduces the notion of a wishlist. In the steps outlined earlier to get the ordered set of features $\mathbb{P}$, we consider one query feature at a time (the last feature appended to $\mathbb{P}$), and survey the original list of features $\mathbb{O}$ to get the feature most correlated with that query feature. When surveying the features in $\mathbb{O}$, we only consider the most correlated features that are not already present in $\mathbb{P}$, while keeping track of the features that would have been selected had they not been duplicates. We save these features as the "wishlist" of the query feature. When all features have been processed, wishlists with more than two features are then considered in descending order based on their size. If a set of wishlists is identified such that all features in $\mathbb{O}$ are part of at least one wishlist, we consider the feature partitioning to be successful, and treat every wishlist as a separate feature group to be received by a separate input layer by a modular MLP.



(a) Baseline MLP



(b) Modular MLP

Figure 4.2: Sketches illustrating the difference between (a) a baseline Multi-Layer Perceptron (MLP), and (b) a modular MLP used by our method. The modular MLP differs mainly in the input layer, that is made up of multiple input layers accepting different feature groups.

In the cases where no feature clusters emerge, we use the ordered set of features $\mathbb{P}$ as an input to a one dimensional CNN. The idea is that the ordered set places features that are related as neighbors, allowing the kernels to learn the entities they constitute or the relationships between them. This rationale is similar to other approaches mentioned earlier (e.g. DeepInsight, REFINED), but our method uses a different relationship metric and performs the feature transformation within the same space (the transformed feature vector is still tabular), rather than transforming the feature vector into the visual space. We argue that the idea of bringing related features to be within enough proximity to be picked up by a convolutional neural network does not necessitate the transformation of the features into pixels and using a 2D CNN, and that our approach makes the learning task more computationally efficient with a smaller memory footprint and a reduced hypothesis space that may facilitate superior optimization of the cost function while requiring less time and resources.

CHAPTER 5
EVALUATING TABULAR FEATURE TRANSFORMATION AND MODULAR LEARNING USING SYNTHETIC EXPERIMENTS

In this chapter, we provide an empirical evaluation of our method presented in Chapter 4. We introduce three experiments (The Hanoi Experiment, The Rectangle Experiment, and The Bit Vector Experiment) that use synthetically generated datasets. These experiments are designed in a way that allows different levels of representation of the entities and relationships that formulate the machine learning problem. This allows us to examine how the learning outcome changes as a result of feature transformation and information representation. The results of these experiments add further evidence supporting our hypothesis stating that highlighting entities and relationships enhances representation learning and yields a better learning outcome using less time and parameters. We also use a one-dimensional version of the Modified National Institute of Standards and Technology (MNIST) dataset [100] to evaluate the version of our approach that shuffles the feature vector without partitioning it.

5.1 The Hanoi Experiment

In The Hanoi Experiment, we use a synthetically generated dataset of ordered numerical sequences of length 10. The sequences are originally sorted in ascending order, before shuffling between 0 and 3 distinct pairs of numbers in the sequence. The number of pairs that has been shuffled is the label assigned to each sequence, making this a multi-class classification problem (either the sequence is still in order, or anywhere between 1 to 3 distinct pairs have been shuffled). We compare the performance of different representations of these sequences using different learners. Visually, the sequence is represented as a vertical stack of rectangles aligned at the center as shown in Figure 5.1. The width of each rectangle represents the corresponding number in the sequence (i.e the top rectangle's width represents the first number in the sequence when scaled according to the entire image width, the second rectangle's width corresponds to the second number of the sequence, and so on). The dataset consists of 10,000 balanced examples, 1,500 of which are used for validation.

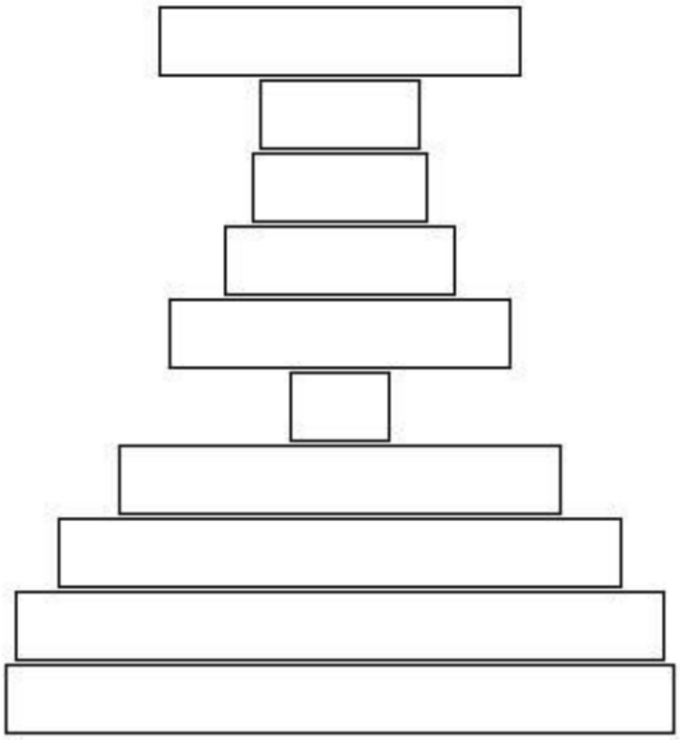


Figure 5.1: An example of images generated for The Hanoi Experiment. Each example is a sequence of numbers originally sorted in ascending order. The task is to classify the number of distinct pairs (between 0 and 3 pairs) that were swapped after generating the sequence. Visually, the sequence is represented as a stack of rectangles where each number is represented by the width of its corresponding rectangle.

We used the FASTAI library to perform these experiments. We used the CNN learner for the visual representation, and the Tabular learner for the tabular representation. The CNN learner used is the ResNet18 model. We experimented with both a pre-trained and un-trained versions of the model. The tabular learner took longer training time to converge, and human time to fine tune the hyper-parameters. We ended up using a network having two hidden layers with 20 neurons each, and a maximum learning rate equal to 0.05. Clearly the two models have a significant difference when it comes to the number of parameters. The difference in accuracy is also noticeable: the tabular learner yielded an 89% testing accuracy, while the ResNet18 model yielded a 97.9% testing accuracy.

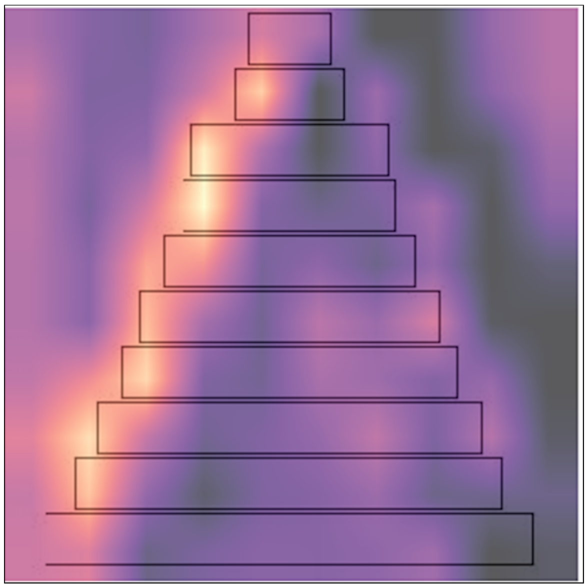
To gain insight into the learning process, we investigated some of the visual learner's confident decisions using heatmaps generated by the GradCam approach mentioned earlier. The gradcam approach highlights the regions in the image that are most influencing the learner's decision. A gallery of the generated heatmaps is shown in Figure 5.2.

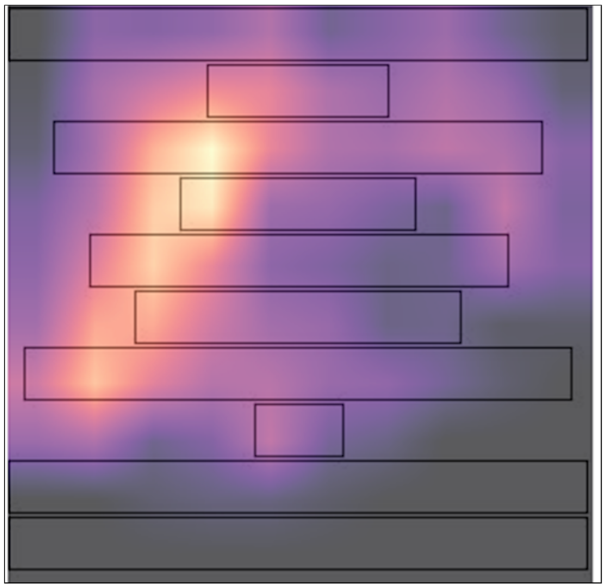
The figures suggest that in a significant number of cases, the learner's attention is at the outer edges of the rectangles, which define the relationships between the numbers, rather than on the widths of the rectangles themselves. To test this idea further, we modified the visual representation to keep the same information (vertical stack of rectangles representing the numbers' widths) while sabotaging the centered alignment that facilitates the learning of relationships between each number in the sequence and the next. To do that, we aligned every other rectangle with the left margin of the image, and the rectangles in between to the right margin, as shown in Figure 5.3.

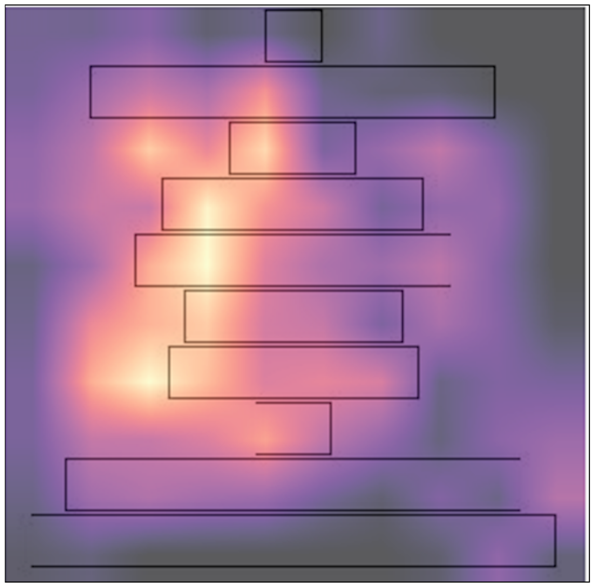
Using this modified representation, we see a significant drop in performance, with the testing accuracy dropping from 97.9% to 85.1%. This adds confidence to our claim that the model is learning the relationships between the features, and that a representation that makes it harder to capture feature relationships, leads to a poor learning outcome compared to a representation that highlights feature relationships, even when using the same learner.

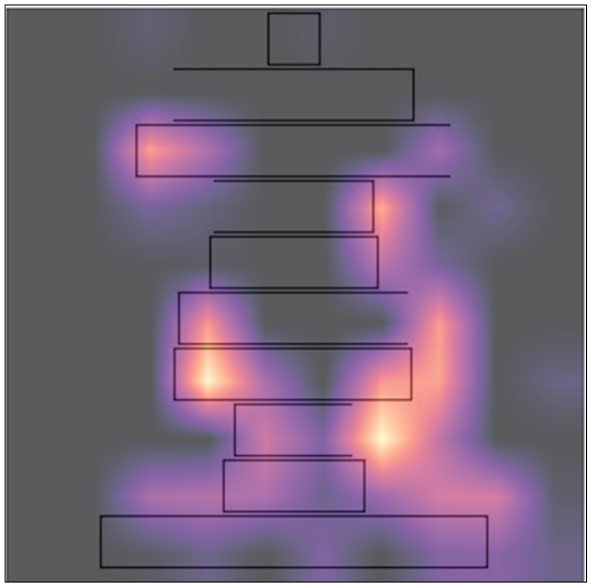
To stretch this idea further, we modified the tabular representation in a way that highlights the relationships between the features. Namely, we appended the tabular input data with a sequence of signs highlighting the difference between consecutive pairs of numbers in the original input sequence (for every ordered pair (x,y) in the sequence, we appended -1 if $x < y$, and +1 otherwise). This improved the performance of the tabular learners significantly, making it on par with the visual learner's performance (97.2%). Table 5.1 summarizes the results of all experiments performed on this dataset. We also include results using a pre-trained version of the ResNet18 model, to highlight the effectiveness of transfer learning, showing an improvement over using a visual learner with random weights.

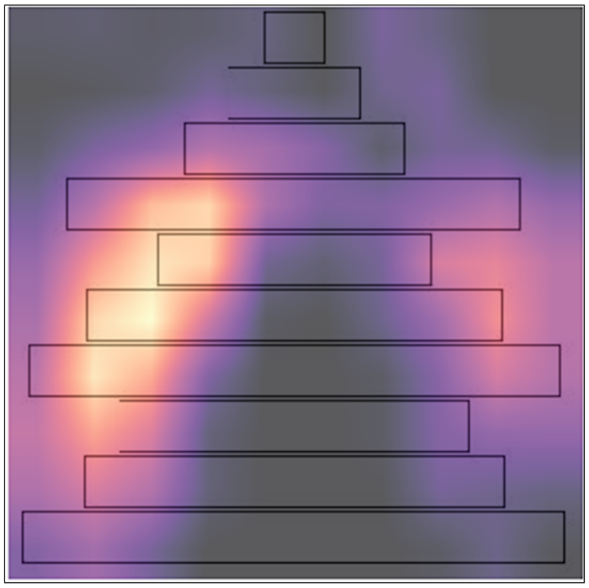
Figure 5.2: Heatmaps generated by the GradCam approach, overlayed over input images used in The Hanoi Experiment. The GradCam approach highlights the regions in the image that are most influencing the model's decision.

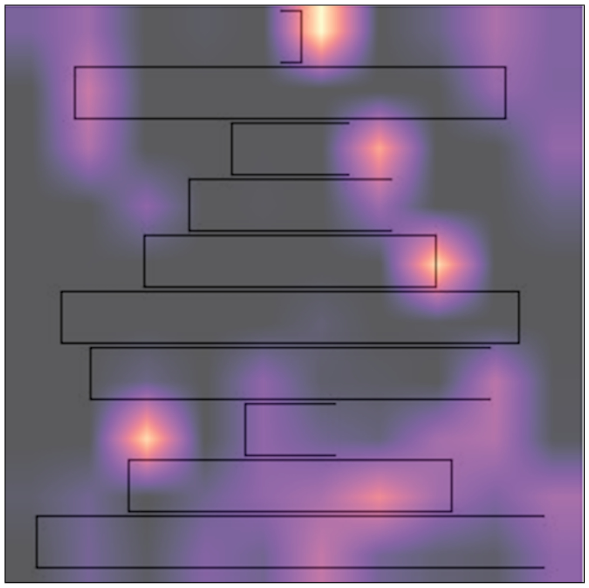

(a) Example 1

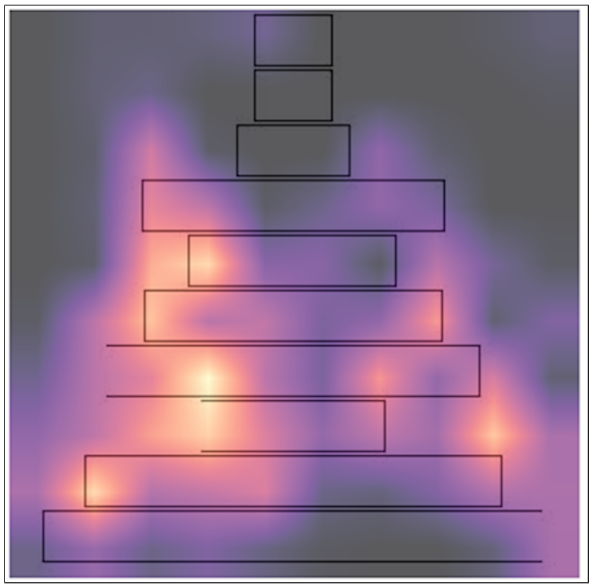

(b) Example 2


(c) Example 3


(d) Example 4


(e) Example 5


(f) Example 6


(g) Example 7

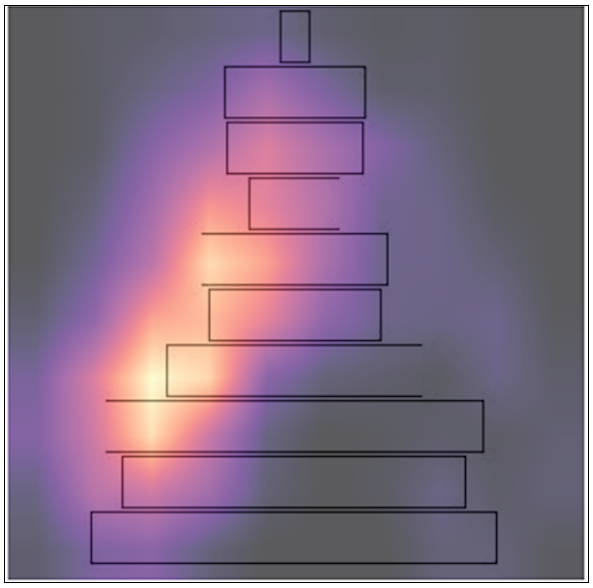
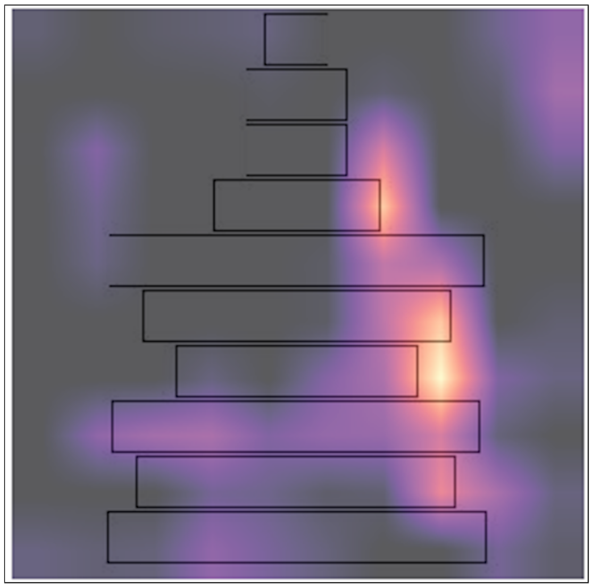
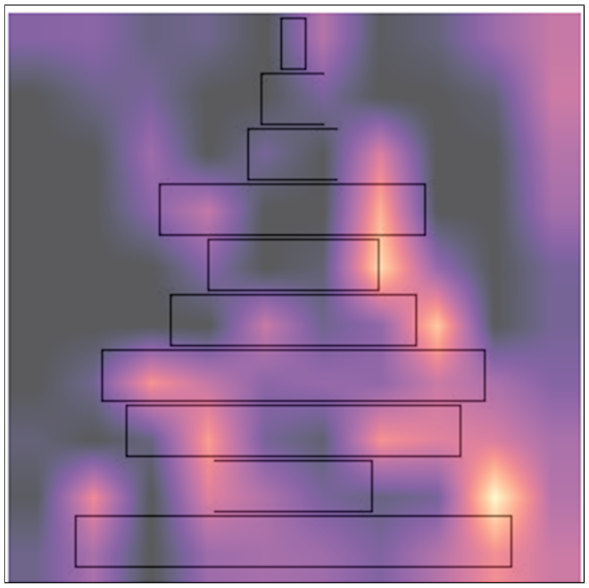
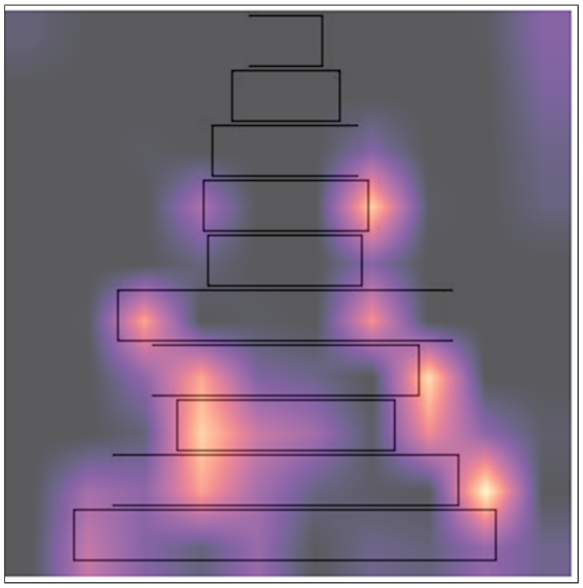

(h) Example 8

Figure 5.2, continued.


(i) Example 9


(j) Example 10


(k) Example 11

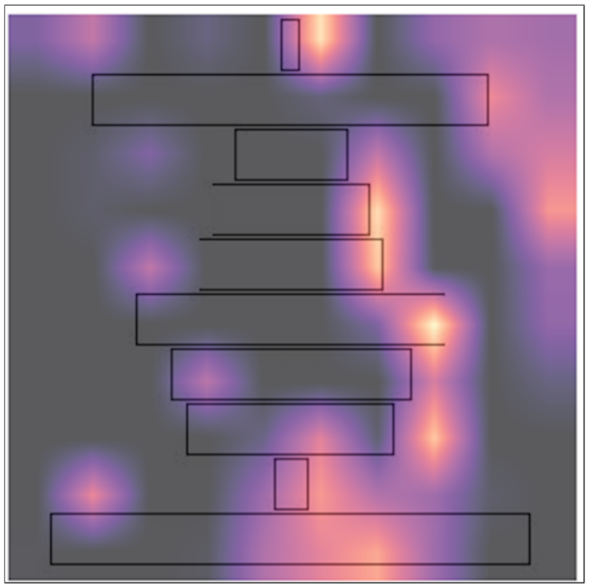
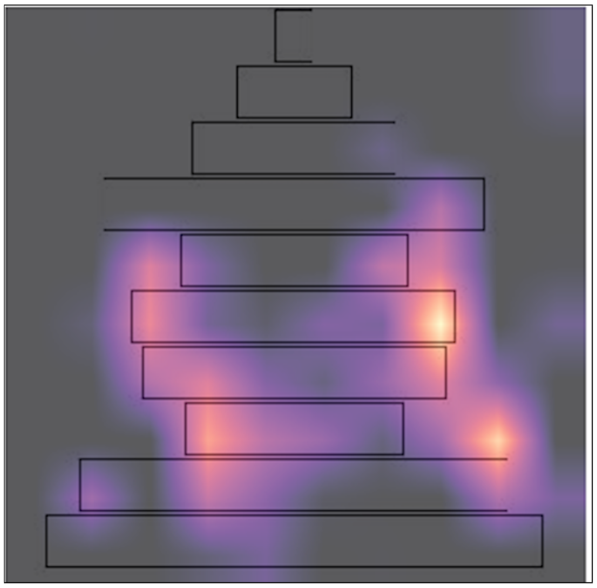
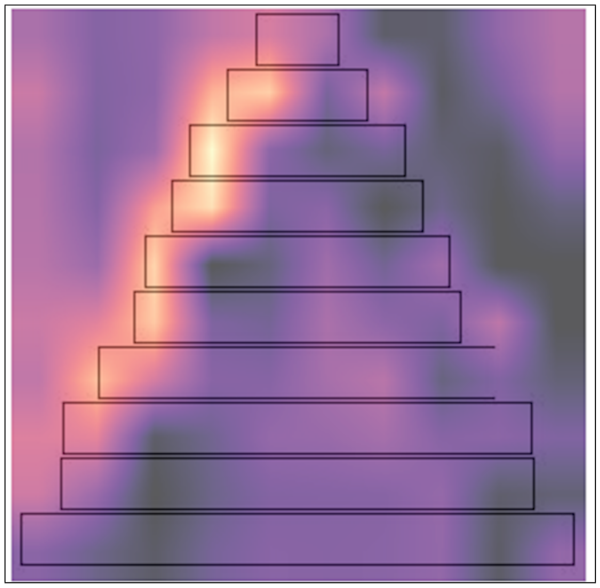
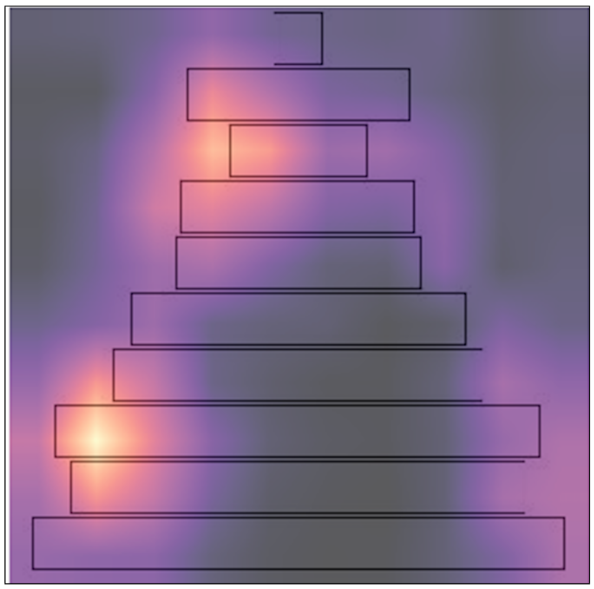

(l) Example 12

Figure 5.2, continued.


(m) Example 13


(n) Example 14


(o) Example 15

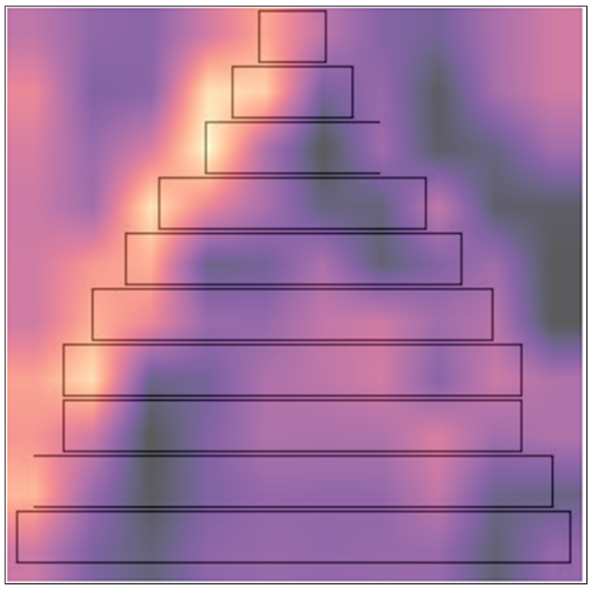

(p) Example 16

Figure 5.2, continued.

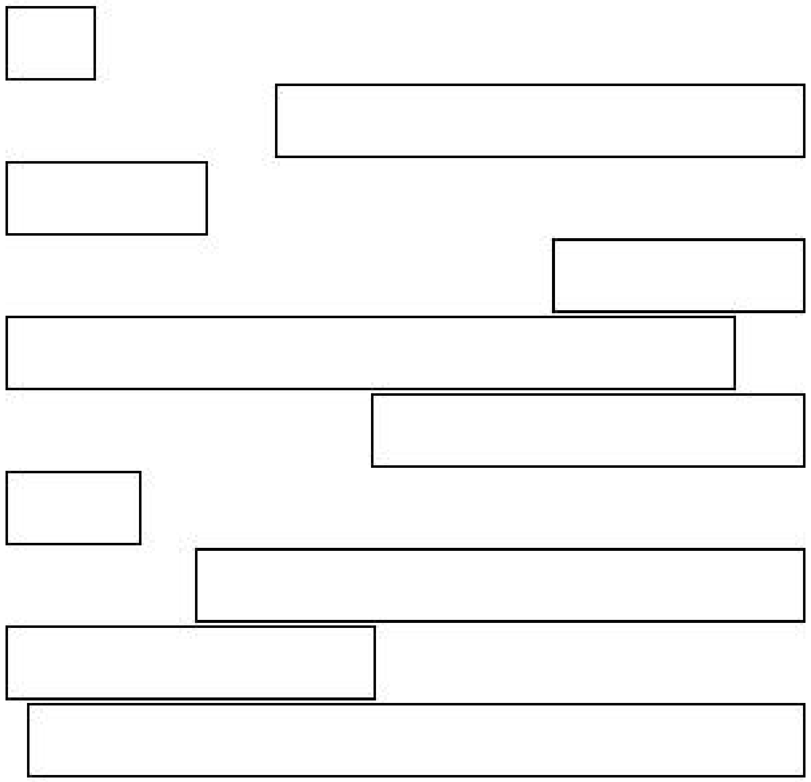


Figure 5.3: A modified visualization of the examples in The Hanoi Experiment. The problem remains the same: predicting the number of distinct pairs swapped in an ordered sequence of numbers represented visually as a vertical stack of rectangles, but the rectangles are now no longer aligned at the center of the image. Instead, the rectangles are left and right justified one at a time (even-indexed rectangles are aligned with the left margin of the image, odd-indexed rectangles are aligned with the right margin of the image).

Table 5.1: Summary of the results observed in The Hanoi Experiment: Testing accuracy of tabular and visual learners (baseline and pre-trained) on different forms of representations of the information. The baseline visual representation transforms every number in the sequence to a rectangle, but does not align the rectangles in any meaningful way. The transformed visual representation aligns the rectangles at the center of the image, facilitating the learning of relationships between them. The baseline tabular representation is the 10 numbers representing the sequence. The transformed tabular representation appends the relationship (smaller/larger) between every pair of numbers to the baseline tabular representation.

Representation	Visual Pre-trained	Visual Baseline	Tabular
Baseline	87%	85%	89%
Transformed	**99%**	**98%**	**97%**

These experiments support our idea that representing the same information in a way that highlights objects and the relationships between them make the problem easier for the learner to learn and may have a significant effect on the learning outcome. The experiments also suggest that certain transformations highlighting the relationships between features or different feature attributes may significantly bridge the performance gap between the two mainstream modes of representation: the tabular and the visual, while maintaining the representation mode.

5.2 The Rectangle Experiment

The Rectangle experiment revolves around the relationship between two objects: a rectangle and a single point. The task is binary classification with the question being whether the point falls inside the rectangle or not. The dataset is synthetically generated and the distribution between the two classes is balanced. It is made up of 10,000 examples, 3,000 of which are used for validation. Both the rectangle and the single point fall in the first quadrant of a Cartesian plane (in the square defined by (0, 0) and (300, 300). The features representing these objects are:

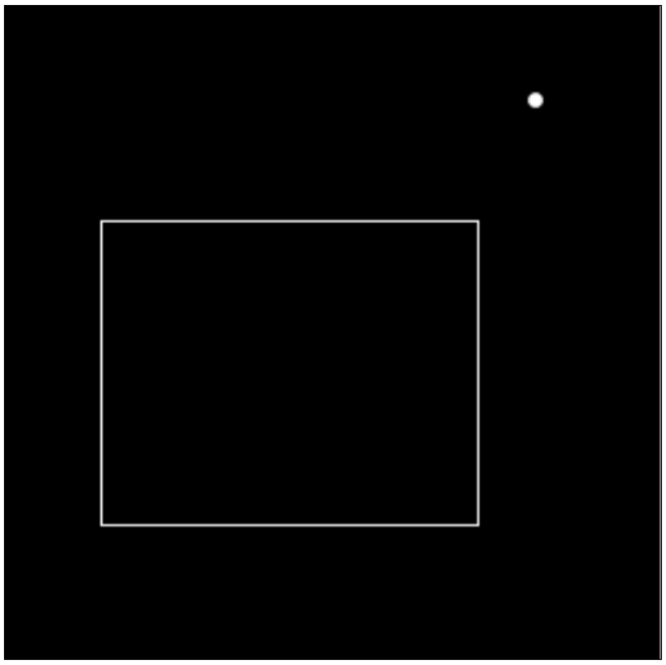
- The 2D coordinates of the four corners of the rectangle, and
- the 2D coordinates of the single point.

We evaluate the performance of different learners on different representations of the same information. In the tabular space, we compare the performance of a baseline MLP to that of a modular MLP. Specifically for this example, the modular MLP accepts the input features belonging to each of the two axes separately. We divide the features across the two axes following our domain knowledge that the function that determines the label (whether the point is inside the rectangle or not) is a conjunction of two functions comparing x-coordinates and y-coordinates separately. We also compare the performance of the tabular learners to a visual learner that takes as input an image representing the rectangle and the point (as a filled circle). We experiment with two forms of visual representation of the rectangle: the first shows the relationship between the points in the feature vector, by connecting the relevant points and drawing the rectangle as a human would. The second is a direct representation of the tabular features that only visualizes the four corner points without the connecting lines. Figure 5.4 shows an example of the different visualizations.

To evaluate the tabular learners, we used Keras running on a Tensorflow backend, with the Sigmoid activation function for the hidden layers, and default settings of the Adam optimizer. We trained both learners for 100 epochs and report the best achieved testing accuracy. The modular MLP scores a testing accuracy of 99.3% compared to 95.6% achieved by the baseline version. Further experiments suggest that the baseline accuracy may increase further if left to train for longer periods of time, but the overall trend is that it'll always be lower (even if slightly) than the modular version. To evaluate the visual learners, We used FastAI with the ResNet18 model, which has substantially more parameters than the tabular counterparts. We train the visual learners for 25 epochs and report the maximum testing accuracy. Table 5.2 summarizes the results achieved by the different learners on the different representations. Note that when using a pre-trained version of the visual learner, the sparse representation's performance improves to 98%, while the dense representation that includes the connecting lines does not change.

Table 5.2: Summary of the results observed in The Rectangle Experiment: Testing accuracy of tabular and visual learners on different forms of representations of the information. The baseline visual representation transforms every coordinate to a single point (filled circle), but does not connect the four corners of the rectangle. The transformed visual representation include the rectangle lines. The baseline tabular representation is the 10 numbers corresponding to the (x, y) coordinates of the 5 points. The transformed tabular representation separates the numbers into two groups: one with the x-coordinates of the points and another with the y-coordinates.

Representation	Tabular	Visual
Baseline	95.6%	96.7%
Transformed	**99.3%**	**99.7%**


(a) Transformed visual representation

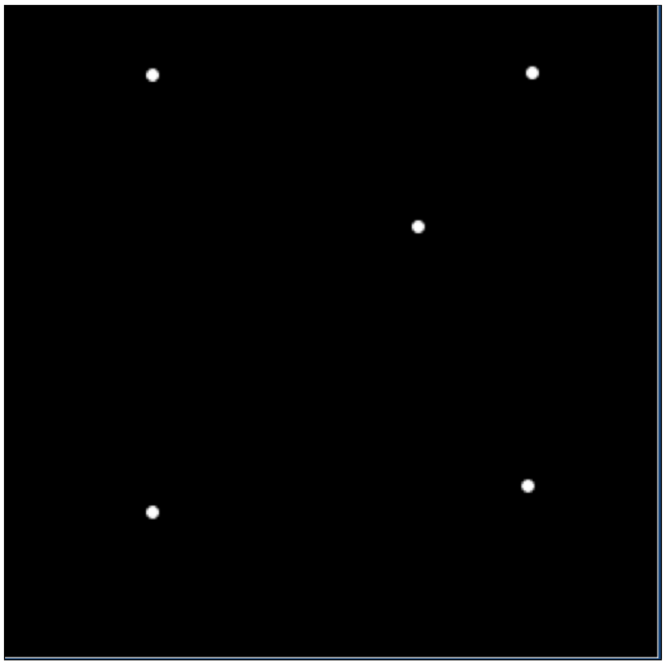

(b) Baseline visual representation

Figure 5.4: An example of images generated for The Rectangle Experiment. The task is to classify whether a point lies inside or outside the rectangle described by four other points. Two representations are used, figure (a) shows a transformed representation where the entities (rectangle corners) are connected with lines, highlighting the relationships between them similar to how a human would, figure (b) shows a baseline visual representation of the raw data with only the 5 points and no connecting lines.

5.3 The Bit Vector Experiment

In The Rectangle Experiment, we know that to ascertain whether the point falls within the bounds of the rectangle we need to determine the truth value of the conjunction of two algebraic functions: one that examines the x-coordinates and another that examines the y-coordinates of the points in the feature vector. Namely, if the rectangle's upper left corner is (x1, y1) and lower right corner is (x2, y2) and the point in question is (q1, q2), then the rule-based approach would ascertain the truth value of (x1 <= q1 <= x2) AND (y1 <= q2 <= q2) to determine whether the point in question falls within the bounds of the rectangle or not. We hypothesized that separating the feature vector across the two axes, in a way that allows the learning of each sub-function separately before learning the conjunction between them, would yield a better learning outcome, and need less time and parameters. While the experiments performed were in line with our expectations, the difference in learning outcome was acute given the little room for improvement over the worst performing learners. In this section, we introduce two more experiments with a higher margin for a performance comparison. Similar to The Rectangle Experiment, these experiments involve a number of sub-functions grouped together by another function. There are 15 features divided equally across 5 groups. Each group is an input to a sub-function, and all 5 sub-functions are an input to a higher-order function. We hypothesize that presenting these features in a way that allows each sub-function to be learned separately before the higher-order function can be learned would lead to quicker learning that needs less parameters, and a better learning outcome.

Our rationale is that in some cases, the rule based methods clearly involve subsets of features interacting amongst themselves (e.g. the two axes in The Rectangle Experiment, or the five sub-functions in The Bit Vector Experiment). In such cases, when the right answer for a given learning instance is determined by a subset of these sub-functions, the features forming the other sub-functions act as noise. To demonstrate this idea, consider the example (10, 10, 60, 10, 10, 60, 60, 60, 30, 100) from the Rectangle experiments. The values correspond to (x1, y1, x2, y2, x3, y3, x4, y4, x5, y5), where (x1,y1)..(x4,y4) are the four corners of the rectangle and (x5, y5) is the point in question. Clearly the point (30, 100) falls outside of the boundaries of the described rectangle due to its y-coordinate. Now assume that all the features are fed through a single input layer to an MLP. For the MLP to match the rule-based function, a subset of the features would need to be positively reinforced (those describing the x-coordinates of the points, since the x-coordinate of the query point is within the bounds of the x-coordinates of the rectangle), but the rest of the features would need to be negatively reinforced (since the y-coordinate of the point is outside the bounds of the y-coordinates of the rectangle). Thus, an artificial neuron connecting all features would receive conflicting feedback, with some features acting as a learning signal and the rest acting as noise. To compensate, the learner might need more time, parameters, or to learn a function that is entirely different from the rule-based one but that scores well on the given dataset.

5.3.1 Evaluating Modular Learning on Conjunction-Based Functions

In the following experiments, the dataset is synthetically generated and spans all possible values for 15-bit vectors. The 15 bits are then divided into 5 equal groups (every 3 consecutive bits), and the label is assigned as the truth value of the resulting Boolean function made up of the conjunction of the bits in each group and the disjunction of the groups.

More formally, let the input vector V = b1,..., b15. F is defined as:

$$f_1 = b_1 \wedge b_2 \wedge b_3 \tag{5.1}$$

$$f_2 = b_4 \wedge b_5 \wedge b_6 \tag{5.2}$$

$$f_3 = b_7 \wedge b_8 \wedge b_9 \tag{5.3}$$

$$f_4 = b_{10} \wedge b_{11} \wedge b_{12} \tag{5.4}$$

$$f_5 = b_{13} \wedge b_{14} \wedge b_{15} \tag{5.5}$$

$$F = f_1 \vee f_2 \vee f_3 \vee f_4 \vee f_5 \tag{5.6}$$

The label of each example then is the truth value of F. The overall dataset is made up of 32,768 examples, 15,961 of which belong to the positive class. We test the performance of different tabular learners on the generated dataset. Specifically, we compare the performance of a modular MLP that accepts the input as five separate groups, to a baseline MLP with a single input layer. The modular MLP has the least number of parameters, 26, which is the bare minimum needed to accept each group of features separately and generate an output. The lower number of parameters suggests a disadvantage when compared to the larger baseline learners, but the modular MLP matches the rule-based accuracy (100%) and is significantly better than a baseline MLP with more parameters (77.3% testing accuracy achieved by a baseline MLP with 35 parameters). We experiment with two different architectures for the baseline MLP learners. Specifically, we use a learner with one hidden layer, similar to the modular learner, but experiment with a different number of neurons in that layer. We also train a learner with two hidden layers. We report the testing performance of the different learners, along with the number of parameters, in Table 5.3.

Table 5.3: Summary of the results observed in The Bit Vector Experiment (Conjunction): the testing accuracy and number of parameters of the different evaluated tabular learners. The baseline tabular learners accept the entire 15 bit vector using a single input layer. The modular tabular learner partitions the input vector into five groups: one for every 3 bits constituting a sub-function, and passes each group through a separate input layer.

	Testing Accuracy	Number of parameters
Modular	**100%**	**26**
Baseline	77.3%	35
Baseline	82.3%	86
Baseline	94.7%	88 (2 layers)
Baseline	99.9%	256

All models were trained for 100 epochs, but the modular model converged around the 60th epoch, quicker than the rest. Examining Table 5.3, an additional benefit of using modular approaches with a smaller number of parameters may lie in restricting the hypothesis space, similar to how CNNs use weight sharing to produce similar effects and correspondingly a superior learning outcome.

5.3.2 Evaluating Modular Learning on Disjunction-Based Functions

In the following experiments, the dataset also spans all possible values for 15-bit vectors, grouped into 5 equal groups (every 3 consecutive bits), but we change the Boolean function F to be the conjunction of the groups, with each group being a disjunction of the bits constituting it.

More formally, let the input vector V = b1,..., b15. F is defined as:

$$f_1 = b_1 \vee b_2 \vee b_3 \tag{5.7}$$

$$f_2 = b_4 \vee b_5 \vee b_6 \tag{5.8}$$

$$f_3 = b_7 \vee b_8 \vee b_9 \tag{5.9}$$

$$f_4 = b_{10} \vee b_{11} \vee b_{12} \tag{5.10}$$

$$f_5 = b_{13} \vee b_{14} \vee b_{15} \tag{5.11}$$

$$F = f_1 \wedge f_2 \wedge f_3 \wedge f_4 \wedge f_5 \tag{5.12}$$

The label of each example then is the truth value of F. The overall dataset is also made up of 32,768 examples, 16,807 of which belong to the positive class.

We hypothesize that the baseline MLP learner would show improved results in this variant of the experiment. Our rationale is that in this variant, less features are needed to determine the right outcome, so in a way, it takes less to be right. Also, if the learner is learning a function that is different from the rule-based one, it is easier to find a function between seemingly un-related features that produce the right output. Thus, the number of parameters out-numbering the number of features in MLPs might prove to be an advantage to a baseline learner in this case.

We also test the performance of different tabular learners on this dataset and report the results, along with the number of parameters of each learner, in Table 5.4.

Table 5.4: Summary of the results observed in The Bit Vector Experiment (Disjunction): the testing accuracy and number of parameters of the different evaluated tabular learners. The baseline tabular learners accept the entire 15 bit vector using a single input layer. The modular tabular learner partitions the input vector into five groups: one for every 3 bits constituting a sub-function, and passes each group through a separate input layer.

	Testing Accuracy	Number of parameters
Modular	**100%**	**26**
Baseline	79.6%	35
Baseline	81.2%	35 (100 epochs)
Baseline	94.9%	86
Baseline	90.9%	88 (2 layers)
Baseline	100%	256

All models were trained for 25 epochs. We also trained the baseline version for 100 epochs (mentioned in Table 5.4). While the experiments prove that the baseline MLP shows an improved performance on this variant of the experiment, it takes a learner with ten times the number of parameters of the modular MLP to match its testing accuracy.

5.4 The Mnist1D Dataset

MNIST1D is a synthetic dataset constructed with the purpose of discriminating machine learning models performance [100]. It is generated using template patterns that represent handwritten digits between 0 and 9, analogous to the original MNIST dataset [101]. The templates are then subjected to random amounts of padding, translation, correlated noise, iid noise, and scaling, leading to a wide range of testing accuracy when tested using key machine learning models, as shown in Table 5.5.

A more challenging version of this dataset shuffles the features before training following a method similar to that described in [102]. The MNIST1D dataset consists of 4,000 training

Table 5.5: Testing accuracy achieved by different models on the MNIST-1D dataset.

Dataset	Logistic regression	MLP	CNN	GRU	Human expert
MNIST-1D	32%	68%	94%	91%	96%

examples and 1,000 testing examples. Each example is represented by 40 features. Figure 5.5 by Sam Greydanus [100] shows a visualization of the tabular features as compared to the original MNIST examples. We used Numpy's random shuffle function to get a shuffled version of the dataset.

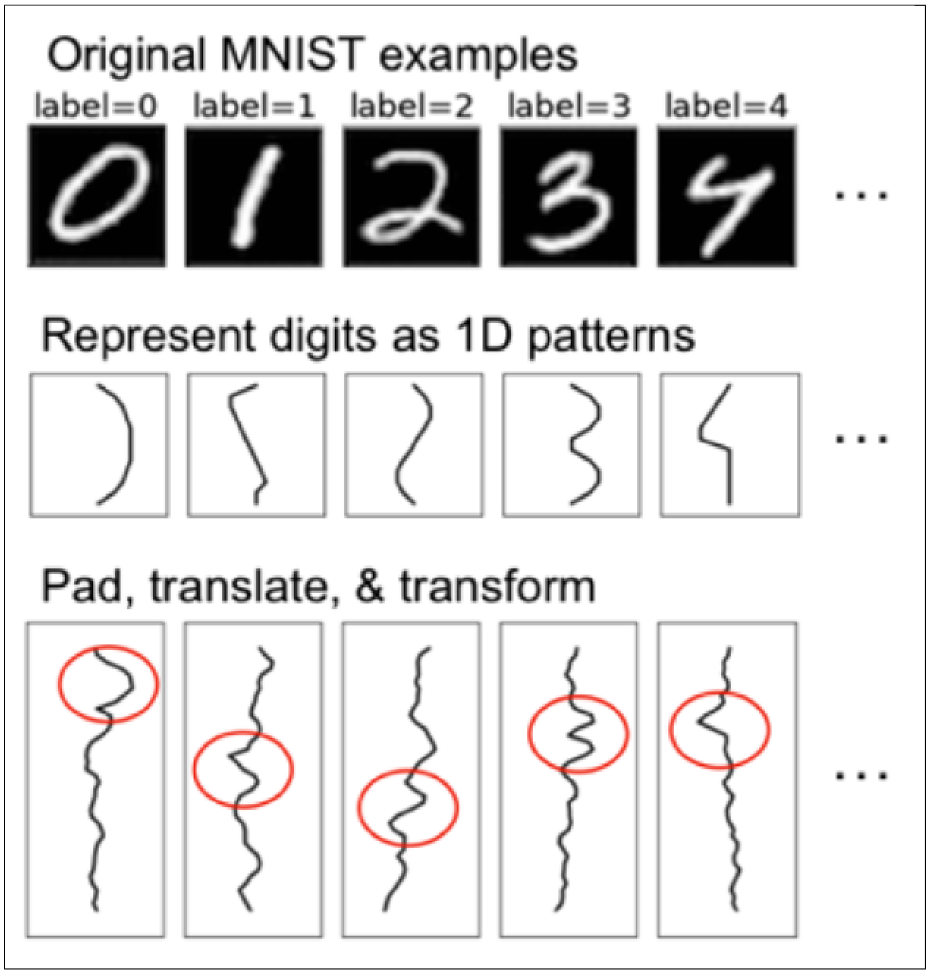


Figure 5.5: Generating the MNIST1D dataset: Digits are represented as one-dimensional patterns that are then padded, translated, and transformed and subject to other processes. In our experiments, we use a shuffled version of this dataset which randomly shuffles the resulting feature vectors. Adapted from (Sam Greydanus 2020 [100])

Our method did not produce any feature clusters, so we resorted to re-ordering the features using their correlation matrix. We present a comparison of the performance of different methods in addition to our method in Table 5.6.

Table 5.6: Testing accuracy achieved by different models on the shuffled version of the MNIST-1D dataset, including our approach (CNN re-ordered) which uses the same CNN model but with a re-ordered input feature vector.

Dataset	Logistic regression	MLP	CNN	GRU	Human expert	CNN-reordered
MNIST-1D	32%	68%	56%	57%	30%	**87%**

The performance of the CNN approach on the un-shuffled version of the data is 91%, so clearly using our approach restores a similar level of performance and outperforms all other methods on the shuffled version of the dataset. For a fair comparison of the results, we use the same convolutional neural network without changing the configuration or any of the hyper-parameters.

CHAPTER 6
CROSS-TOOL COMPARISON AND EVALUATION OF TABULAR FEATURE TRANSFORMATION AND MODULAR LEARNING USING REAL WORLD EXPERIMENTS

In this chapter, we compare our approach presented in Chapter 4, to some of the other approaches automating feature transformation and selection mentioned in that chapter. We resort to real world datasets that were used by the other methods to validate their results. We especially focus on the experiments that are not performance saturated, and that offer enough margin for a meaningful comparison. We use the same experimental setup devised by the other tools. We thoroughly checked their specifications (and where possible, code-bases) to devise a similar setup. Details about the data and the model hyper-parameters are provided at the end of the chapter.

6.1 Comparison to the TabNet Method

TabNet uses a feature transformer and an attentive transformer as part of its architecture to select the most salient features for a given learning instance. The goal is to have a reduced model capacity that is efficient to train but can effectively learn by directing its learning power to the selected features. We compare our method to TabNet on datasets used to validate their results. Our approach outperforms TabNet on all the datasets reported in this section. The models we used are modular MLPs with an architecture described in Chapter 4, and details provided at the end of this chapter.

6.1.1 The Poker Hand Dataset

The Poker Hand Dataset represents a multi-class classification problem with the goal of classifying Poker hands [103]. Each example is a Poker hand consisting of five cards drawn from a 52-card deck. The cards are represented by two features: the card's suit and rank. Thus, each learning instance in the dataset is represented by a total of ten features capturing the five (suit, rank) pairs corresponding to the five cards. The output is one of the ten possible Poker hands. The class distribution in the dataset is heavily imbalanced. Table 6.1 shows a breakdown of the number of examples across the ten possible classes.

Table 6.1: Poker Hands representation in the training set: the percentage of examples in the training set representing the corresponding poker hand. These percentages are an approximate representation of the actual possible distributions of poker hands in a 52-card deck.

Poker Hand	Representation
High Card	49.952%
One Pair	42.379%
Two Pairs	4.822%
Three of a Kind	2.051%
Straight	0.372%
Flush	0.216%
Full House	0.144%
Four of a Kind	0.024%
Straight Flush	0.020%
Royal Flush	0.020%

Conventional methods suffer from the severe imbalance undermining the data. Using our approach, the features were clustered into two groups: one containing the card ranks and another containing the card suits. Such a breakdown could be expected given the different range of values allowed for the ranks (1 to 13 ranks) and suits (1 to 4 suits), which would affect the correlation scores between the features. The breakdown also matches our domain knowledge of the problem, since the resulting Poker hand is usually a function of either the ranks (e.g One Pair, Two Pairs, Three of a Kind), the suits (e.g. Flush), or a combination of both (Straight Flush). We compare the performance of different methods reported in TabNet to our approach in Table 6.2. Our method outperforms all others and almost matches the rule-based method performance.

Note that while the result tables in TabNet report a testing accuracy of 50% for a baseline MLP, our experiments yielded different results, with a 99.2% testing accuracy, which

Table 6.2: Testing accuracy achieved by different models evaluated using the Poker dataset.

Model	Accuracy
Decision Tree (DT)	50%
Baseline MLP	50%
Deep Neural DT	65.1%
XGBoost	71.1%
LightGBM	70%
CatBoost	66.6%
TabNet	99.2%
Modular MLP	**99.9%**

is significantly higher than what is reported in the TabNet study, but still lower than the results achieved using our modular approach. The performance achieved by our approach is still significantly better than TabNet and the other approaches given the highly imbalanced nature of the dataset (six of the ten classes correspond to roughly 0.8% of the class distribution).

6.1.2 The Higgs Boson Dataset

The Higgs boson dataset introduces a binary classification problem with the goal of "distinguishing a signal process that produces Higgs bosons from a background process that does not" [104]. Each example in the dataset is represented by 28 features, 21 of which are low-level features representing the momentum and other traits of particles generated after a collision event, which are common to both the signal and the background process. The remaining 7 features are functions of the 21 low-level ones defined with domain knowledge about the intermediate processes separating the signal and the background. Our method clustered the features into three groups.

The performance of different machine learning methods evaluated on this dataset falls within a narrow range, with improvements reported by each method being extremely minute. Similar to TabNet, we experimented with different neural network sizes. We compare the results achieved by our method to different machine learning models in Table 6.3. Note that the small version of our model outperforms nearly all other methods that have significantly more parameters.

Table 6.3: Testing accuracy along with the number of parameters corresponding to the different models evaluated using the Higgs Boson dataset.

Model	Test Accuracy	Model Size
Sparse Evolutionary MLP	78.47	81k
Gradient boosted tree-S	74.22	0.12M
Gradient boosted tree-M	75.97	0.69M
MLP	78.44	2.04M
Gradient boosted tree-L	76.98	6.96M
Tabnet-S	78.25	81K
Tabnet-M	78.84	0.66M
Modular MLP-S	**78.74**	**81K**
Modular MLP-M	**79.04**	**0.41M**

6.1.3 The Sarcos Dataset

The Sarcos dataset introduces a multi-variate regression task with the goal of predicting the inverse dynamics of an anthropomorphic robot arm given its position, velocity, and acceleration values [105]. Each of these values is represented by 7 features, leading to an input feature vector with 21 features. The arm has seven degrees of freedom making this a multi-variate regression task. The dataset is part of the validation experiments reported by TabNet. In our approach, we feed each of the feature groups separately into an input layer. We compare the performance of our approach to the methods mentioned in TabNet in Table 6.4. Our approach achieves significantly better than the other methods, using much less parameters.

To validate our results further, we trained a baseline MLP with the same number of parameters used in the small modular MLP, and achieved a score of 1.19, which is significantly better than the score reported for the MLP in TabNet using much more parameters, and outperforms almost all the other methods mentioned in Table 6.4, with the exception of the modular MLP and the larger versions of Tabnet.

Table 6.4: Testing accuracy along with the number of parameters corresponding to the different models evaluated using the Sarcos dataset.

Model	Test MSE	Model Size
Random forest	2.39	16.7k
Stochastic decision tree	2.11	28K
MLP	2.13	0.14M
Adaptive neural tree	1.23	0.6M
Gradient boosted tree	1.44	0.99M
Tabnet-S	1.25	6.3K
Tabnet-M	0.28	0.59M
Tabnet-L	0.14	1.75M
Modular MLP-S	**1.15**	**6.5K**
Modular MLP-M	**0.09**	**0.27M**

6.2 Comparison to the DeepInsight Method

As mentioned in Chapter 4, DeepInsight transforms tabular features into images by placing related features as neighboring pixels. The performance of DeepInsight is evaluated on five datasets. One of these datasets required domain knowledge by the authors, which we did not include in our evaluations. We compare the performance of our method to that of DeepInsight and other tabular learners and present the results in Table 6.5. Our approach outperforms DeepInsight on three of the datasets, and matches its performance on a performance-saturated one.

The following is a brief description of the datasets reported by DeepInsight that we evaluate our method on:

- RELATHE - is a textual dataset extracted from a larger newsgroup documents with the binary classification task of predicting the newsgroup each document belongs to [106]. It is made up of 1,427 examples each represented by 4,322 features.
- RNA-seq is a public gene expression dataset extracted from TCGA (https://cancergenome.nih.gov). It contains 6,216 examples each represented by 60,483 features. It is a multi-class classification problem with 10 possible cancer types.
- Madelon - is a highly non-linear synthetic dataset introduced in the NIPS 2003 feature

extraction challenge and represents a binary classification problem. It contains 2,600 examples each represented by 500 features.

- Ringnorm-DELVE is another synthetic dataset that implements the Leo Breiman's ringnorm example [107] and represents a binary classification problem. It contains 7,400 examples each represnted by 20 features.

Our method did not yield any feature clusters. Thus, the feature transformation applied is a re-ordering of the original feature vector based on their correlation scores. Our intuition is similar to that applied by DeepInsight, but we use a different relationship metric and apply the transformation within the same space (the resulting feature vector is still tabular rather than visual). To exploit local relationships between the re-ordered features, we use a one dimensional convolutional neural network as a tabular learner. More details about the network architecture and hyper-parameters can be found at the end of this chapter.

Table 6.5: Testing accuracy achieved by the different models (DT stands for Decision Trees) evaluated using two datasets (Relathe and Madelon) reported in the DeepInsight study, as compared to a baseline one-dimensional convolutional neural network and our approach which re-ordered the input feature vector (re-ordered 1D CNN).

Dataset	DT	Ada-Boost	Random Forests	DeepInsight	1D CNN (Baseline)	1D CNN (re-ordered)
Relathe	87%	85%	90%	92%	94%	**96%**
Madelon	65%	60%	62%	88%	92%	**94%**

In Table 6.5, we focused on the datasets that are not performance saturated and leave enough room for comparison. The other two datastes are reported in the DeepInsight study with an accuracy of 98% and 99% (corresponding to Ringnorm-DELVE and RNA-Seq respectively). Our method achieved an accuracy of 99% for both of these datasets. Also, while in the original study, the results achieved by DeepInsight are not compared to any other artificial neural networks, we report the performance of the same convolutional neural networks used by our approach, but without the feature re-ordering step performed by our method. These are listed as baseline 1D CNN in Table 6.5. Note that these learners achieve a higher performance than all other learners, including DeepInsight, but lower than our method.

6.3 Comparison to the REFINED Method

As mentioned in Chapter 4, REFINED is another method that transforms tabular features into images. It uses the euclidean distance between features to determine the corresponding pixel locations, with the idea of representing related features as neighboring pixels. One of the experiments they validate their results on uses the Genomics of Drug Sensitivity in Cancer (GDSC) dataset [108], "which describes the responses to 222 anticancer drugs across approximately 972 cancer cell lines". The cell lines are represented by 1,211 gene expression values, and the drugs are represented by 992 chemical descriptors. The goal is to predict the drug response as measured by the IC50 value, which captures the concentration of drugs needed to reduce the response by 50%, making this a regression task. We use a modular MLP by presenting each of the drugs and the cell lines separately using two input layers. Note that this is similar to the REFINED approach, where they pass two separate images representing the drugs and the cell lines through two input arms to a two-dimensional convolutional neural network. We attribute the superior performance of our method to the simpler neural network architecture and the smaller resulting hypothesis space.

Note that when performing their experiments, they resort to an 80-10-10 train-validate-test split, and cite computational limitations as a factor hindering more experimentation and hyper-parameter fine tuning, and evaluating their results via cross-validation. Thus, an additional advantage of our method is using smaller models with data that is significantly less memory-intensive and more computationally efficient. We perform k-fold cross-validation, training on the same percentage as their method, but testing on all remaining folds. For example, when we train our method on 20% on the data, we test on the remaining 80%. Using these significantly larger testing datasets adds confidence to the results reported by our method. More importantly, the results emphasize an advantage of our method when it comes to the amount of data needed to train a model. When training our model using

20% of the training dataset and testing it on the remaining 80%, the results achieved are better than all other models (except REFINED) trained on 50% of the dataset and tested on 10%. When training our model on 50% of the dataset and testing on the remaining 50%, the results achieved are better than all models (and similar to REFINED) trained on 80% of the data and tested on 10%. Table 6.6 shows a comparison of our method to those reported in the REFINED study.

We also evaluated our method using 10-fold cross validation to compare the performance on a testing set of a similar size to the ones achieved via their 80-10-10 split. The results achieved by our modular MLP were 0.389 NRMSE and 0.921 PCC, which are further improvements in favor of our method.

6.4 Data and Hyper-Parameters

We used Keras running on a Tensorflow backend, and unless otherwise stated, the default settings of the Adam optimizer. Random seeds were fixed for reproducibility. We reported the best testing accuracy achieved by our method while training the model.

6.4.1 The Poker Hand Dataset

The Poker Hand Dataset consists of 25,010 training examples and 1,000,000 testing examples. Each example is represented by 10 features. In the modular approach, the input features are split into two equal groups and fed separately through two input layers. Both tabular learners we experimented with had two hidden layers, with the baseline MLP having more parameters than the modular one. Each input layer in the modular learner was connected to 128 neurons, which were concatenated before being connected to another hidden layer with 16 neurons. The difference in the baseline learner is that its single input layer was connected to a hidden layer with 256 neurons. We used the Sigmoid activation function in the hidden layers followed by Softmax activation for the output layer.

Table 6.6: Normalized root-mean-square error (NRMSE) and Pearson correlation coefficient (PCC) achieved by different models evaluated on the Genomics of Drug Sensitivity in Cancer (GDSC) dataset and reported in the REFINED study, compared to our approach (Modular MLP). The training percentage resembles the percentage of the data each model was trained on. Note that with the exception of our Modular MLP, all models were evaluated on a 10% split from the dataset, whereas the results reported for our Modular MLP are the cross-validation score while training on the reported percentage and evaluating our model on the rest of the data.

Model	Trained on 20%		Training on 50%		Training on 80%	
	NRMSE	PCC	NRMSE	PCC	NRMSE	PCC
EN	0.890	0.488	0.889	0.484	0.887	0.486
RF	0.609	0.797	0.620	0.785	0.569	0.821
SVR	0.750	0.847	0.742	0.845	0.525	0.853
ANN	1.407	0.519	0.475	0.883	0.435	0.901
Random CNN	0.579	0.836	0.456	0.892	0.441	0.903
PCA CNN	0.612	0.820	0.461	0.891	0.443	0.901
REFINED CNN	0.541	0.845	0.439	0.899	0.414	0.911
Modular MLP	**0.454**	**0.891**	**0.414**	**0.911**	**0.393**	**0.920**

6.4.2 The Higgs Boson Dataset

The Higgs Boson Dataset consists of 11,000,000 examples the last 500,000 of which are used for testing. Each example in the data is represented by 28 features. Our method separated the features into three clusters containing 17, 6, and 5 features each. The three groups of features were fed to the modular learner through three separate input layers. Our modular MLP had two hidden layers. Each input layer was connected to 256 neurons, which were then concatenated to form the first hidden layer. The concatenated layer was then connected with another hidden layer having 95 neurons. We used the ReLU activation function for the hidden layers and the Sigmoid activation function for the output layer, with Amsgrad enabled and a learning rate of 0.001.

6.4.3 The Sarcos Dataset

The Sarcos dataset consists of 48,933 examples, 4,449 of which are used for testing. Each example is represented by 21 features. The features describe 7 joint position (7 features), velocity (7 features), and acceleration (7 features) values of an anthropomorphic robot arm with 7 degrees of freedom. We trained two modular MLPs of varying sizes (modular MLP-S and modular MLP-M, referring to a small and a medium sized modular MLP learners). The input was passed through three separate input layers, each including the 7 features corresponding to one of the variables mentioned earlier (i.e position, velocity, or acceleration). In the modular MLP-S network, each input layer was connected with a hidden layer containing 28 neurons. The three resulting layers were then concatenated and connected to another hidden layer with 64 neurons. In the modular MLP-M network, the three input layers were connected to a hidden layer with 128 neurons each. The three resulting layers were then concatenated and connected to another layer with 512 neurons, which was connected to another layer with 128 neurons, which was connected to a final hidden layer with 32 neurons. We also trained a small baseline MLP network (baseline MLP-S), which received the input as one input layer connected to a layer with 70 neurons, which in turn was connected to another layer with 64 neurons. The Sigmoid function was used as an activation function for all hidden layers, and the linear function was used for the output layer.

6.4.4 The DeepInsight Experiments

We used four datasets mentioned in the DeepInsight study to evaluate and compare the performance of our method: RNA-seq, Relathe, Madelon, and Ringnorm-DELVE. Similar to DeepInsight's validation approach, we followed an 80-10-10 training-validation-testing split. Our approach did not produce feature clusters but re-ordered the features based on their correlation matrix. The re-ordered feature vector was then used as input to a 1D CNN with two hidden layers using the Sigmoid activation function. For the Relathe dataset, the first hidden layer was made of 64 kernels of width 128, and the second layer was made of 8 kernels of width 3. For the Madelon dataset, the first layer consisted of 16 kernels of width 50, connected to a second layer made of 8 kernels of width 5. For the Ringnorm-DELVE dataset, the first layer was made of 16 kernels of width 5, connected to a second layer made of 8 kernels of width 2. For the performance-saturated RNA-seq data, we simply used a tabular learner from the FASTAI API with a single hidden learner having 30,000 neurons.

6.4.5 The REFINED Experiment

To compare our method to the REFINED study, we used the GDSC dataset that is part of their validation experiments. The dataset involves the interaction between 972 cell lines and 222 drugs. Each cell line is represented by 1,211 features, and each drug by 972 features. Our modular approach uses a neural network with two hidden layers. First, the cell lines and the drugs are passed separately through two input layers connected to 1,024 neurons. These neurons are then concatenated and connected with another hidden layer with 128 neurons. We used the ReLU activation function for the input layers and the Linear activation function for the output layer.

CHAPTER 7
SUMMARY AND FUTURE DIRECTIONS

We demonstrated that representing information in a way that highlights entities and relationships facilitates the machine learning process and yields a better learning outcome. We presented feature transformation strategies towards that end, and resorted to empirical evidence using synthetic and real world datasets for evaluation. The feature transformations are either driven by human domain knowledge, such as by representing tabular features visually using manually crafted designs (e.g. representing genes as arrows in the problems of identifying genomic islands and detecting operons in bacterial genomes presented in chapters 2 and 3), or by partitioning the input features into groups, where each represents a separate entity or relationship. The different representations are used as input to different machine learners with different inductive biases. For the visual representations, a two-dimensional convolutional neural network is used. For the partitioned representation, a modular MLP with multiple input layers is used. In cases were human domain knowledge is lacking, the feature transformation is a permutation of the feature vector such that the re-ordered version places related features within close proximity. For such representations, the re-ordered feature vector is used as input to a one-dimensional convolutional neural network. This is in agreement with the recent trend underpinning the approaches presented in Chapter 4, that automatically transform tabular data into images with the goal of representing related features as neighboring pixels to be learned from by the convolution operator in 2D CNNs. We showed that such automatic transformations to the visual domain are unnecessary, and that applying a similar approach of re-ordering the features and using a 1D CNN is not only sufficient but also achieved better results.

In all these approaches (except manually engineered visual representations), the underlying idea is having certain feature groups act as single units. Our intuition behind that is that for certain problems, some features should be considered together to determine the quality of the learner's predictions, while the rest of the features might act as noise dampening the learning signal. For example, in certain classification tasks (e.g. The Rectangle Experiment in Chapter 5), certain features could satisfy a condition that positively reinforces the learner's output, while other features negatively re-inforce the same predicted output. The corresponding output and error being back-propagated would have to update the weights attached to all these features in the case of a fully connected baseline MLP, since all the features would be connected to the same neurons. For that purpose, fully connected MLPs have the a lower signal to noise ratio than networks that limit the connections. Thus, MLPs may need more training time, data, or parameters to compensate. The limited connections in alternative learners cannot be random and have to be meaningful in exploiting properties inherent in the data. CNNs work because of spatial locality and corresponding properties assumed in images. This is the main motivation behind representing related features as neighboring pixels and using 2D CNNs in the approaches mentioned in Chapter 4, and our approach that partitions the feature vector and feeds separate partitions through separate input layers to a modular MLP (or a re-ordered feature vector to a 1D CNN).

Examining the modular MLP we presented earlier, one can see that it could be achieved starting with a baseline MLP by means of skipping (or selectively deleting) certain connections (Figure 4.2). Our next main objective is to automate this process and make it part of the learner, to become part of the representation learning process rather than a pre-processing transformation step. The outcome would be a method whereby the data is presented to a network similar to the baseline model, but with a new layer type that knows how to modulate the architecture and find meaningful feature partitions, for example by using gated layers or winner-take-all layers. The goal then is to embed the feature transformations as part of the learner, leading to an end-to-end process that allows the learner to perform representation learning using raw data and without any pre-processing or human engineering. As a weaker goal, a simpler way to automate the approach in its current form is by using something like Keras pre-processing layers. While this would not make the feature transformations part of the learning process, it provides an end-to-end model that performs the transformations before training.

We mention some potential improvements that could be applied to the current version of our method: while we used Pearson's Correlation Coefficient to automate the feature re-ordering and partitioning, we believe that a survey of different statistical measures (e.g. Spearman's Rho, tree based methods or machine learning methods with intrinsic feature selection, information gain) and their properties could offer some insight into the limitations and applicability of these measures in different domains and on different types of datasets and problems. Moreover, just like feature importance is an established concept, feature group (or feature unit) importance could be useful, especially for guiding a network architecture design that allows different feature groups with different levels of importance to be represented differently (e.g. using more neurons/layers), which might add further benefits to a learner that divides and directs its learning capacity unequally across the feature groups.

Another major future direction is to investigate the different modes of learning used by the baseline and the modular MLPs. For example in The Bit Vector Experiments, we could see how different types of functions (conjunction vs disjunction based functions) benefit at different levels from partitioning the input feature vector and using a modular MLP. This raises the question of whether different classes of functions exist, and whether they can be identified automatically. In case different classes of functions are identified, another question is whether the effectiveness of the different approaches presented in this work can be measured and matched to these different classes.

In addition to achieving a better learning outcome, a major contribution from seeking answers to some of the questions presented in this work could be a better understanding of the learning process. Our approach inspires some experiments that could shed light and allow us to gain a better insight about the machine learning behavior. For example, it would be interesting to show that the modular MLP learns a function closer to the rule based method than the non-modular MLP that needs more parameters to emulate that function, by dampening some neuron weights and amplifying others to achieve a good testing score on the same dataset. We believe that any contribution to the theoretical foundation around the ideas discussed in this work and the theory of learning in general, would be most valuable.

REFERENCES

[1] Goyal, Anirudh, and Yoshua Bengio. "Inductive biases for deep learning of higher-level cognition." arXiv preprint arXiv:2011.15091 (2020).

[2] Kuhn, Max, and Kjell Johnson. Applied predictive modeling. Vol. 26. New York: Springer, 2013.

[3] Assaf, Rida, Fangfang Xia, and Rick Stevens. "Identifying genomic islands with deep neural networks." BMC genomics 22, no. 3 (2021): 1-11.

[4] Dobrindt, U., Hochhut, B., Hentschel, U., Hacker, J. (2004) Genomic islands in pathogenic and environmental microorganisms. *Nat. Rev. Microbiol.*, **2**, 414—424.

[5] da Silva Filho, Antonio Camilo, Roberto Tadeu Raittz, Dieval Guizelini, Camilla Reginatto De Pierri, Diônata Willian Augusto, Izabella Castilhos Ribeiro dos Santos-Weiss, and Jeroniza Nunes Marchaukoski. "Comparative analysis of genomic island prediction tools." Frontiers in genetics 9 (2018): 619.

[6] Langille, M., Hsiao, W., Brinkman, F. (2010) Detecting genomic islands using bioinformatics approaches. *Nature Reviews Microbiology*, **8(5)**, 373–382.

[7] Hacker, J., et al. (1990) Deletions of chromosomal regions coding for fimbriae and hemolysins occur in vitro and in vivo in various extraintestinal Escherichia coli isolates. *Microb. Pathog*, **8**, 213—225.

[8] Hudson, C., Lau, B., Williams, K. (2014) Islander: a database of precisely mapped genomic islands in tRNA and tmRNA genes. *Nucleic Acids Research*, **43(D1)**, D48–D53.

[9] Barondess, J.J., Beckwith, J. (1990) A bacterial virulence determinant encoded by lysogenic coliphage lambda. *Nature*, **346**, 871—874.

[10] Lu, B., Leong, H. (2016) Computational methods for predicting genomic islands in microbial genomes. *Computational And Structural Biotechnology Journal*, **14**, 200–206.

[11] Juhas, M., van der Meer, J.R., Gaillard, M., Hood, D.W., et al. (2009) Genomic islands: tools of bacterial horizontal gene transfer and evolution. *FEMS Microbiol. Rev.*, **33**, 376—3793.

[12] Akhter, S., Aziz, R., Edwards, R. (2012) PhiSpy: a novel algorithm for finding prophages in bacterial genomes that combines similarity- and composition-based strategies. *Nucleic Acids Research*, **40(16)**, e126–e126.

[13] Fogg, P., Colloms, S., Rosser, S., Stark, M., Smith, M. (2014) New applications for phage integrases. *Journal of Molecular Biology*, **426(15)**, 2703–2716.

[14] Hambly, E., Suttle, C.A.. (2005) The viriosphere, diversity, and genetic exchange within phage communities. *Curr. Opin. Microbiol.*, **8**, 444-–50.

[15] Hacker, J., Kaper, J.B. (2000) Pathogenicity islands and the evolution of microbes. *Annu. Rev. Microbiol.*, **54**, 641—679.

[16] Choi, I.G., Kim, S.H., (2007) Global extent of horizontal gene transfer. *PNAS*, **104(11)**, 4489–4494.

[17] Arndt, D., Grant, J., Marcu, A., Sajed, T., Pon, A., Liang, Y., & Wishart, D. (2016) PHASTER: a better, faster version of the PHAST phage search tool. *Nucleic Acids Research*, **44(W1)**, W16–W21.

[18] Coates, A.R., Hu, Y. (2007) Novel approaches to developing new antibiotics for bacterial infections. *Br. J. Pharmacol.*, **152**, 1147—1154.

[19] Bar, H., Yacoby, I., Benhar,I. (2008) Killing cancer cells by targeted drug-carrying phage nanomedicines. *BMC Biotechnol.*, **8**, 37.

[20] Hacker, J., Blum-Oehler, G., Muhldorfer, I., Tschape, H. (1997) Pathogenicity islands of virulent bacteria: structure, function and impact on microbial evolution. *Mol. Microbiol.*, **23**, 1089—1097.

[21] Schmidt H, Hensel M. (2004) Pathogenicity Islands in bacterial pathogenesis. *Clin. Mcrobiolog. Rev.*, **17**, 14—56.

[22] Ho Sui, S.J,, Fedynak, A., Hsiao, W.W.L., Langille, M.G.I., Brinkman, F.S.L. (2009) The association of virulence factors with genomic islands. *PLoS One*, **4**, e8094.

[23] Moriel, D.G., Bertoldi, I., Spagnuolo, A., Marchi, S., Rosini, R., et al. (2010) Identification of protective and broadly conserved vaccine antigens from the genome of extraintestinal pathogenic Escherichia coli. *Proc. Natl. Acad. Sci. U S A*, **107**, 9072-–9077.

[24] Langille, M.G., Hsiao, W.W., Brinkman, FS. (2008) Evaluation of genomic island predictors using a comparative genomics approach. *BMC Bioinformatics*, **9**, 329.

[25] Srividhya, K.V., Rao, G.V., Raghavenderan, L., Mehta, P., Prilusky,J., Manicka,S., Sussman, J.L., Krishnaswamy, S. (2006) Database and comparative identification of prophages. In: Huang,D-S, Li,K and Irwin,GW (eds). *Intelligent Control and Automation, Lecture Notes in Control and Information Sciences.* Springer, Berlin, Vol. 344, pp. 863—868.

[26] Ester, M., Kriegel, H., Sander, J., Xu, X. (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: *KDD-1996 Proceedings* AAAI Press, Menlo Park, pp. 226—231.

[27] Hsiao, W., Wan, I., Jones, S.J., et al. (2003) IslandPath: aiding detection of genomic islands in prokaryotes. *Bioinformatics*, **19(3)**, b418—420.

[28] Waack, S., Keller, O. Asper, R., et al. (2006) Score-based prediction of genomic islands in prokaryotic genomes using hidden Markov models. *BMC Bioinformatics*, **7**, 142.

[29] Tu, Q., Ding, D. (2003) Detecting pathogenicity islands and anomalous gene clusters by iterative discriminant analysis. *FEMS Microbiol. Lett.*, **221**, 269—275.

[30] Vernikos, G. S., Parkhill, J. (2006) Interpolated variable order motifs for identification of horizontally acquired DNA: revisiting the Salmonella pathogenicity islands. *Bioinformatics*, **22**, 2196—2203.

[31] Fouts, D. (2006) Phage_Finder: automated identification and classification of prophage regions in complete bacterial genome sequences. *Nucleic Acids Res.*, **34**, 5839-–5851.

[32] Langille, M.G., Brinkman, F. IslandViewer: an integrated interface for computational identification and visualization of genomic islands. *Bioinformatics*, **25**, 664-–665.

[33] Nelson, K.E., Weinel, C., Paulsen, I.T., Dodson, R.J., Hilbert, H., Martins dos Santos, V.A., Fouts, D.E., Gill, S.R., Pop, M., Holmes, M. et al. (2002) Complete genome sequence and comparative analysis of the metabolically versatile Pseudomonas putida KT2440. *Environ. Microbiol.*, **4**, 799—808.

[34] Zhang, R., Zhang, C.T. (2004) A systematic method to identify genomic islands and its applications in analyzing the genomes of Corynebacterium glutamicum and Vibrio vulnificus CMCP6 chromosome I. *Bioinformatics*, **20(5)**, 612—622.

[35] Wattam, A.R, ZDavis, J.J., Assaf, R., Boisvert, S., Bun, T., Conrad, N., Dietrich, E.M., Disz, T., Gabbard, J.L., Gerdes, S., Henry, C.S., Kenyon, R.W., Machi, D., Mao, C., Nordberg, E.K., Olsen, G.J., Murphy-Olson, D.E., Olson, R., Overbeek, R., Parrello, B., Pusch, G.D., Shukla, M., Vonstein, V., Warren, A., Xia, F., Yoo, H., Stevens, R.L. (2017) Improvements to PATRIC, the all-bacterial Bioinformatics Database and Analysis Resource Center. *Nucleic Acids Research* **45(D1)**, D535–D542.

[36] Vernikos, G. S., Parkhill, J. (2008) Resolving the structural features of genomic islands: a machine learning approach. *Genome Res.*, **18**, 331—342.

[37] Karlin, S., Mrazek, J. & Campbell, A. M. (1998) Codon usages in different gene classes of the Escherichia coli genome. *Mol. Microbiol.*, **29**, 1341—1355.

[38] Sandberg, R., et al. (2001) Capturing whole-genome characteristics in short sequences using a naive Bayesian classifier. *Genome Res.*, **11**, 1404—1409.

[39] Hatfull, G.F., Jacobs-Sera, D., Lawrence, J.G., Pope, W.H., Russell, D.A., Ko, C.C., Weber, R.J., Patel, M.C., Germane, K.L., Edgar, R.H., et al. (2010) Comparative genomic analysis of 60 mycobacteriophage genomes: genome clustering, gene acquisition, and gene size. *J. Mol. Biol.*, **397**, 119—143.

[40] Williams, K.P. (2002) Integration sites for genetic elements in prokaryotic tRNA and tmRNA genes: sublocation preference of integrase subfamilies. *Nucleic Acids Res.*, **30**, 866-–875.

[41] Reiter, W.D., Palm, P., Yeats, S. (1989) Transfer RNA genes frequently serve as integration sites for prokaryotic genetic elements. *Nucleic Acids Research*, **17**, 1907-–1914.

[42] Bellanger, X., Payot, S., Leblond-Bourget, N., Guedon, G. (2014) Conjugative and mobilizable genomic islands in bacteria: evolution and diversity. *FEMS Microbiol. Rev.*, **38**, 720—760.

[43] How to Retrain an Image Classifier for New Categories - TensorFlow Hub — TensorFlow. (2018) Retrieved from https://www.tensorflow.org/hub/tutorials/image_retraining

[44] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.. (2015) ImageNet large scale visual recognition challenge. *IJCV*.

[45] Jia, Y., Weiss, R.J., Biadsy, F., Macherey, W., Johnson, M., Chen, Z. Wu, Y., (2019). Direct speech-to-speech translation with a sequence-to-sequence model. arXiv preprint arXiv:1904.06037.

[46] Poplin, R., Chang, P., Alexander, D., Schwartz, S., Colthurst, T., & Ku, A. et al. (2018) A universal SNP and small-indel variant caller using deep neural networks. *Nature Biotechnology*

[47] Howard, J. (2019) Lesson 2: Deep Learning 2019 - Data cleaning and production; SGD from scratch. Retrieved from https://www.youtube.com/watch?v=ccMHJeQU4Qw

[48] Assaf, Rida, Fangfang Xia, and Rick Stevens. "Detecting operons in bacterial genomes via visual representation learning." Scientific Reports 11, no. 1 (2021): 1-10.

[49] Parker, Nina, Mark Schneegurt, Anh-Hue Thi Tu, Brian M. Foster, and Philip Lister. "Microbiology (OpenStax)." (2016).

[50] Fran B, Perrin D, Monod J, et al. The operon : a group of genes whose expression is coordinated by an operator. *J Bacteriol.* **29**, 1727–9 (1960).

[51] Romero,P.R. and Karp,P.D. Using functional and organizational information to improve genome-wide computational prediction of transcription units on pathway-genome databases. *Bioinformatics.* **20**, 709–717 (2004).

[52] Mao, Xizeng, Qin Ma, Chuan Zhou, Xin Chen, Hanyuan Zhang, Jincai Yang, Fenglou Mao, Wei Lai, and Ying Xu. Door 2.0: presenting operons and their functions through dynamic and integrated views. *Nucleic acids research.* **42**, D654-D659 (2014).

[53] Taboada, B., Verde, C., & Merino, E. High accuracy operon prediction method based on STRING database scores. *Nucleic acids research.* **38 (12)**, e130-e130 (2010).

[54] Taboada, B., Ciria, R., Martinez-Guerrero, C. E., & Merino, E. ProOpDB: Prokaryotic Operon Data Base. *Nucleic acids research.* **40 (D1)**, D627-D631 (2011).

[55] Bergman, N. H., Passalacqua, K. D., Hanna, P. C., & Qin, Z. S. Operon prediction for sequenced bacterial genomes without experimental information. *Appl. Environ. Microbiol.* **73 (3)**, 846-854 (2007).

[56] Fortino, V., Smolander, O. P., Auvinen, P., Tagliaferri, R., & Greco, D. Transcriptome dynamics-based operon prediction in prokaryotes. *BMC bioinformatics.* **15 (1)**, 145 (2014).

[57] Hodgman TC. A historical perspective on gene/protein functional assignment. *Bioinformatics.* **16**, 10–5 (2000).

[58] Joon M, Bhatia S, Pasricha R, et al. Functional analysis of an intergenic non-coding sequence within mce1 operon of M. tu- berculosis. *BMC Microbiol.* **10**, 128 (2010).

[59] Wang S, Wang Y, Liang Y, et al. A multi-approaches-guided genetic algorithm with application to operon prediction. *Artif Intell Med.* **41**, 151–9 (2007).

[60] Pantosti A, Sanchini A, Monaco M. Mechanisms of antibiotic resistance in Staphylococcus aureus. *Future Microbiol.* **2**, 323–34 (2007).

[61] Yada,T., Nakao,M., Totoki,Y. and Nakai,K. Modeling and predicting transcriptional units of Escherichia coli genes using hidden Markov models. *Bioinformatics.* **15**, 987–993 (1999).

[62] Craven,M., Page,D., Shavlik,J., Bockhorst,J. and Glasner,J. A probabilistic learning approach to whole-genome operon pre- diction. *Proc. Conf. Intell. Syst. Mol. Biol.* **8**, 116–127 (2000).

[63] Tjaden,B., Haynor,D.R., Stolyar,S., Rosenow,C. and Kolker,E. Identifying operons and untranslated regions of transcripts using Escherichia coli RNA expression analysis. *Bioinformatics.* **18 (Suppl. 1)**, S337–S344 (2002).

[64] Ermolaeva,M.D., White,O. and Salzberg,S.L. Prediction of operons in microbial genomes. *Nucleic Acids Res.* **29**, 1216–1221 (2001).

[65] Zheng,Y., Szustakowski,J.D., Fortnow,L., Roberts,R.J. and Kasif,S. Computational identification of operons in microbial genomes. *Genome Res.* **12**, 1221–1230 (2002).

[66] Okuda S, Kawashima S, Kobayashi K, et al. Characterization of relationships between transcriptional units and operon structures in Bacillus subtilis and Escherichia coli. *BMC Genomics*. **8**, 48 (2007).

[67] Chen,X., Su,Z., Xu,Y. and Jiang,T. Computational prediction of operons in Synechococcus sp. WH8102. *Genome Inform.* **15**, 211–222 (2004).

[68] Tran,T.T., Dam,P., Su,Z., Poole,F.L., Adams,M.W., Zhou,G.T. and Xu,Y. Operon prediction in Pyrococcus furiosus. *Nucleic Acids Res.* **35**, 11–20 (2007).

[69] Dam,P., Olman,V., Harris,K., Su,Z. and Xu,Y. Operon prediction using both genome-specific and general genomic information. *Nucleic Acids Res.* **35**, 288–298 (2007).

[70] Zhang,G.Q., Cao,Z.W., Luo,Q.M., Cai,Y.D. and Li,Y.X. Operon prediction based on SVM. *Comput. Biol. Chem.* **30**, 233–240 (2006).

[71] Bockhorst,J., Craven,M., Page,D., Shavlik,J. and Glasner,J. A Bayesian network approach to operon prediction. *Bioinformatics*. **19**, 1227–1235 (2003).

[72] Edwards,M.T., Rison,S.C., Stoker,N.G. and Wernisch,L. A universally applicable method of operon map prediction on minimally annotated genomes using conserved genomic context. *Nucleic Acids Res.* **33**, 3253–3262 (2005).

[73] Westover,B.P., Buhler,J.D., Sonnenburg,J.L. and Gordon,J.I. Operon prediction without a training set. *Bioinformatics*. **21**, 880–888 (2005).

[74] Jacob,E., Sasikumar,R. and Nair,K.N. A fuzzy guided genetic algorithm for operon prediction. *Bioinformatics*. **21**, 1403–1407 (2005).

[75] Salgado,H., Moreno-Hagelsieb,G., Smith,T.F. and Collado- Vides,J. Operons in Escherichia coli: genomic analyses and predictions. *Proc. Natl Acad. Sci. USA*. **97**, 6652–6657 (2000).

[76] Yan,Y. and Moult,J. Detection of operons. *Proteins.* **64**, 615–628 (2006).

[77] Overbeek, R., M. Fonstein, M. D'Souza, G. D. Pusch, and N. Maltsev. The use of gene clusters to infer functional coupling. *Proc. Natl. Acad. Sci. USA.* **96**, 2896–2901 (1999).

[78] Zaidi, S. S. A., & Zhang, X. Computational operon prediction in whole-genomes and metagenomes. *Briefings in functional genomics.* **16 (4)**, 181-193 (2016).

[79] Szklarczyk, Damian, John H. Morris, Helen Cook, Michael Kuhn, Stefan Wyder, Milan Simonovic, Alberto Santos et al. The STRING database in 2017: quality-controlled protein–protein association networks, made broadly accessible. *Nucleic acids research.* **gkw937** (2016).

[80] Taboada, B., Estrada, K., Ciria, R., & Merino, E. Operon-mapper: a web server for precise operon identification in bacterial and archaeal genomes. *Bioinformatics.* **34 (23)**, 4118-4120 (2018).

[81] Brouwer RWW, Kuipers OP, Van Hijum SAFT. The relative value of operon predictions. *Brief Bioinform.* 367–75 (2008).

[82] Santos-Zavaleta, Alberto, Heladia Salgado, Socorro Gama-Castro, Mishael Sánchez-Pérez, Laura Gómez-Romero, Daniela Ledezma-Tejeida, Jair Santiago García-Sotelo et al. RegulonDB v 10.5: tackling challenges to unify classic and high throughput knowledge of gene regulation in E. coli K-12. *Nucleic acids research.* **47, no. D1**, D212-D220. (2019)

[83] Sierro N., Makita Y., de Hoon M.J.L. and Nakai K. DBTBS: a database of transcriptional regulation in Bacillus subtilis containing upstream intergenic conservation information. *Nucleic Acids Res.* **36 (Database issue)**, D93-D96; (2008)

[84] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. *In Proceedings of the IEEE international conference on computer vision.* 618-626 (2017).

[85] Okuda, S. and Yoshizawa, A.C. ODB: a database for operon organizations, 2011 update. *Nucleic Acids Res.* **39 (Database issue)**, D552-555 (2011).

[86] Davis JJ, Gerdes S, Olsen GJ, Olson R, Pusch GD, Shukla M, Vonstein V, Wattam AR and Yoo H PATtyFams: Protein Families for the Microbial Genomes in the PATRIC Database. *Front. Microbiol.* 7-118. (2016)

[87] FastAI — FastAI. Retrieved from *https://docs.fast.ai/index.html* (2018)

[88] Sharma, Alok, Edwin Vans, Daichi Shigemizu, Keith A. Boroevich, and Tatsuhiko Tsunoda. "DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture." *Scientific reports* 9, no. 1 (2019): 1-7.

[89] Van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9, no. 11 (2008).

[90] Bazgir, Omid, Ruibo Zhang, Saugato Rahman Dhruba, Raziur Rahman, Souparno Ghosh, and Ranadip Pal. "Representation of features as images with neighborhood dependencies for compatibility with convolutional neural networks." Nature communications 11, no. 1 (2020): 1-13.

[91] Zhu, Yitan, Thomas Brettin, Fangfang Xia, Alexander Partin, Maulik Shukla, Hyunseung Yoo, Yvonne A. Evrard, James H. Doroshow, and Rick L. Stevens. "Converting tabular data into images for deep learning with convolutional neural networks." Scientific reports 11, no. 1 (2021): 1-11.

[92] Han, Huimei, Ying Li, and Xingquan Zhu. "Convolutional neural network learning for generic data classification." Information Sciences 477 (2019): 448-465.

[93] Ghoshal, Torumoy. "An Experimental Study on the Applicability of Convolutional Neural Networks Beyond Image Data." PhD diss., The University of Mississippi, 2020.

[94] Ke, Guolin, Jia Zhang, Zhenhui Xu, Jiang Bian, and Tie-Yan Liu. "TabNN: A universal neural network solution for tabular data." (2018).

[95] Ucar, Talip, Ehsan Hajiramezanali, and Lindsay Edwards. "SubTab: Subsetting Features of Tabular Data for Self-Supervised Representation Learning." Advances in Neural Information Processing Systems 34 (2021).

[96] Song, Weiping, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. "Autoint: Automatic feature interaction learning via self-attentive neural networks." In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 1161-1170. 2019.

[97] Golinko, Eric, Thomas Sonderman, and Xingquan Zhu. "Learning convolutional neural networks from ordered features of generic data." In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 897-900. IEEE, 2018.

[98] Kadra, Arlind, Marius Lindauer, Frank Hutter, and Josif Grabocka. "Well-tuned Simple Nets Excel on Tabular Datasets." Advances in Neural Information Processing Systems 34 (2021).

[99] Arık, Sercan O., and Tomas Pfister. "Tabnet: Attentive interpretable tabular learning." In AAAI, vol. 35, pp. 6679-6687. 2021.

[100] Greydanus, Sam. "Scaling down deep learning." arXiv preprint arXiv:2011.14439 (2020).

[101] Deng, Li. "The mnist database of handwritten digit images for machine learning research [best of the web]." IEEE signal processing magazine 29, no. 6 (2012): 141-142.

[102] Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. "Understanding deep learning (still) requires rethinking generalization." Communications of the ACM 64, no. 3 (2021): 107-115.

[103] Dua, Dheeru, and Casey Graff. "UCI machine learning repository." (2017). URL http://archive.ics.uci.edu/ml.

[104] Baldi, P., P. Sadowski, and D. Whiteson. "Searching for Exotic Particles in High-energy Physics with Deep Learning." Nature Communications 5 (July 2, 2014).

[105] Vijayakumar, Sethu, and Stefan Schaal. "Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space." In Proceedings of the seventeenth international conference on machine learning (ICML 2000), vol. 1, pp. 288-293. Morgan Kaufmann, 2000.

[106] Mitchell, Tom M. "Machine learning." (1997).

[107] Breiman, Leo. Bias, variance, and arcing classifiers. Tech. Rep. 460, Statistics Department, University of California, Berkeley, CA, USA, 1996.

[108] Yang, Wanjuan, Jorge Soares, Patricia Greninger, Elena J. Edelman, Howard Lightfoot, Simon Forbes, Nidhi Bindal et al. "Genomics of Drug Sensitivity in Cancer (GDSC): a resource for therapeutic biomarker discovery in cancer cells." Nucleic acids research 41, no. D1 (2012): D955-D961.