University of Ljubljana
Faculty of Computer and Information Science

Domen Mohorčič

Transfer Learning for Phenotype Prediction from Small Gene Expression Data Sets

MASTER'S THESIS

THE 2nd CYCLE MASTER'S STUDY PROGRAMME
COMPUTER AND INFORMATION SCIENCE
Track: Data Science

Supervisor: Prof. Dr. Blaž Zupan

Ljubljana, 2024

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Domen Mohorčič

Učenje s prenosom pri napovedovanju fenotipa na majhnih podatkovnih naborih o izraženosti genov

magistrsko delo

magistrski študijski program druge stopnje
računalništvo in informatika
Smer: Podatkovne Vede

Mentor: Prof. Dr. Blaž Zupan

Ljubljana, 2024

Acknowledgments

I would like to thank my mentor for the constructive feedback I got along the way of creating this thesis. I would also like to thank to my girlfriend Nika and my family for supporting me on my journey through the master's programme. And finally, I would like to thank everyone who contributed with ideas and helped me brainstorm different solutions.

Domen Mohorčič, 2024

Contents

Abstract

Title: Transfer Learning for Phenotype Prediction from Small Gene Expression Data Sets

Recent advances in biotechnology have enabled researchers to collect huge amounts of data, such as gene expression profiles from patients, which provide a foundation for personalized medicine. Such an approach requires the use of machine learning, however, a significant limitation of many medical studies is the small sample size, typically having only a few hundred patients with tens of thousands of features. In this thesis, we addressed this issue by combining multiple small gene expression data sets into a larger one, regardless of the study type, and training deep learning models capable of producing informative gene expression encodings. We used transfer learning to predict the phenotypes on unseen data sets based on the created encodings. We experimented with two model architectures: autoencoders and multi-task models. Although training multi-task models proved challenging, they achieved higher average results on test data sets than autoencoders but never surpassed the results of logistic regression. An examination of the encodings revealed that autoencoders maintained the original data structure whereas the multi-task models mixed samples from different studies, but both proved that the gene expression profile can be reduced to a few informative markers.

Keywords

gene expression, small data set problem, transfer learning, autoencoders, multi-task models

Povzetek

Naslov: Učenje s prenosom pri napovedovanju fenotipa na majhnih podatkovnih naborih o izraženosti genov

Nedavni napredki na področju biotehnologije so raziskovalcem omogočili zbiranje velikih količin podatkov, kot so profili genskih izrazov bolnikov, ki so osnova za personalizirano medicino. Takšen pristop zahteva uporabo strojnega učenja, vendar je glavna omejitev številnih študij majhnost vzorca, ki ima običajno nekaj sto bolnikov z več deset tisoč atributi. V magistrskem delu smo se tega problema lotili tako, da smo združili veliko majhnih podatkovnih naborov o izraženosti genov v en večji nabor in naučili globoke nevronske mreže, zmožne informativnega kodiranja vhodnih podatkov. Uporabili smo učenje s prenosom za napovedovanje fenotipa na kodiranih podatkih iz testnih naborov. Eksperimentirali smo z dvema arhitekturama modelov: samokodirniki in večopravilnimi modeli. Čeprav je bilo učenje večopravilnih modelov zahtevno, so na testnih podatkovnih naborih v povprečju dosegli višje rezultate kot samokodirniki, vendar niso presegli rezultatov logistične regresije. Pri pregledovanju kodiranih vrednosti se je izkazalo, da samokodirniki ohranijo prvotno strukturo podatkov, medtem ko večopravilni modeli ne razlikujejo med primeri iz različnih študij, obe arhitekturi pa sta dokazali, da je profil genskih izrazov možno predstavili le z nekaj vrednostmi.

Ključne besede

genski izrazi, problem majhnih podatkovnih naborov, učenje s prenosom, samokodirniki, večopravilni modeli

Razširjeni povzetek

Razumevanje bioloških podatkov je ključnega pomena v biomedicinskih raziskavah. Ker je običajno podatkov zelo veliko, si raziskovalci pogosto pomagajo s strojnim učenjem [1], s katerim odkrivajo genske kandidate za bolezni [2], raziskujejo interakcije med proteini [3] in iščejo rakave celice [4].

Genski podatki predstavljajo najbolj osnovne biološke podatke. Njihova funkcija in struktura sta del deoksiribonukleinske kisline (DNA) ali ribonukleinske kisline (RNA), kjer je gen definiran kot odsek s specifično funkcijo. DNA se nato prepiše v informacijsko RNA (mRNA), ki jo ribosomi prevedejo v polipeptidno verigo, imenovano protein. Proteini so razlog za delovanje celic in posledično celotnega organizma in imajo tako neposreden vpliv na fenotip organizma, ki je zbirka vseh vidnih lastnosti, kar vključuje izgled, razvoj in biokemične lastnosti.

Zgoraj navedeni postopek opisuje izražanje gena in je neposredna povezava med genotipom in fenotipom. Ta proces nadzoujejo določeni geni [5] in drugi dejavniki [6]. Merjenje izraženosti genov je zelo pomembno za razvoj biomedicinske industrije [7], idealni merski postopek pa bi vključeval detekcijo in štetje proteinov, za katere geni nosijo informacijo, kar je zelo zahtevno in drago. Raziskovalci so zato odkrili posredne metode za merjenje izraženosti genov, ena izmed njih je merjenje števila mRNA v celicah, ki nosijo informacijo o določenih proteinih.

Obstaja veliko javnih podatkovnih zbirk z različnimi biološkimi podatki. Vse vsebujejo ogromne količine podatkov, vsaka pa ima tudi veliko število študij z zelo malo vzorci. Tak primer je podatkovna zbirka ArrayExpress [8],

ki ima več kot 35.000 študij z manj kot 10 primeri. Majhno število vzorcev med biološkimi podatki ni nič nenavadnega, saj je podatke pogosto težko dobiti. Dober primer je Hutchinson-Gilfordov sindrom progerije [9], ki je izjemno redka genetska bolezen. Trenutno je na vsem svetu znanih samo 144 primerov. Učenje globokih modelov na malo primerih je težavno, rešuje se s podvajanjem, interpoliranjem in sintetičnim generiranjem novih primerov.

V tem magistrskem delu bomo problem malega števila primerov reševali z združevanjem več podatkovnih naborov v večjega in z učenjem kodirnega modela, ki bo vhod ustrezno zakodiral in pripravil na napovedovanje fenotipa. Naša hipoteza je, da bo imel tak model večjo možnost za uspeh pri generiranju dobrega latentnega prostora, preverili pa jo bomo z učenjem s prenosom, kjer bomo na kodiranih primerih naučili nove modele in preverili njihovo uspešnost v primerjavi z logistično regresijo, naučeno na nekodiranih primerih.

I Kratek pregled sorodnih del

Napovedovanje fenotipa iz genotipa je kompleksen problem, saj na fenotip vplivajo tudi zunanji dejavniki. Leta 2002 so Beer in ostali ugotovili, da ne morejo natančno napovedati napredovanja pljučnega raka samo s pregledovanjem tkiva [10]. Uspešno so uporabili genske izraze za napovedovanje poteka bolezni in ustvarili seznam genov, s katerimi je možno identificirati kritične bolnike. Leta 2006 so Baty in ostali dokazali, da je na izraženost genov v krvi možno vplivati z različnimi pijačami [11]. Udeležencem so dali grozdni sok, rdeče vino, razredčen etanol in vodo, vzemali krvne vzorce naslednjih dvanajst ur in pokazali, da so se genski izrazi med raziskavo spremenili. Leta 2012 so Schramm in ostali uporabili izraze genov za napovedovanje izida pri udeležencih z rakom, ki vpliva na živčevje [12]. Uspešno so napovedali delež oseb, ki so prebolele raka, in delež tistih, pri katerih se je bolezen ponovila, ter dokazali, da so bili genski izrazi edini pomembni podatek pri napovedovanju. Leta 2020 so Part in ostali uporabili globoke nevronske mreže za diagnosticiranje Alzheimerjeve bolezni [13]. Uporabili so podatke o izraženosti genov in metilaciji DNA ter pokazali, da njihov model klasificira Alzheimerjeve bolnike bolje kot drugi algoritmi.

Učenje s prenosom [14] je metoda strojnega učenja, kjer že naučeni model uporabimo na drugi domeni. Učenje s prenosom se pogosto uporablja v domenah z ogromno podatki, kjer je učenje od začetka nesmiselno, te domene pa so slike [15] in naravni jezik [16]. V genetiki se učenje s prenosom pogosto uporablja kot demonstracija pridobljenega znanja iz enega organizma na drugem organizmu.

Choi in ostali so raziskali uporabo kodiranja genskih izrazov z nevronskimi mrežami [17]. Njihov pristop je bil zelo podoben matrični faktorizaciji, pokazali pa so, da je model podobne vrste raka uvrstil blizu v latentni prostor. Du in ostali so razvili metodo gene2vec za kodiranje genov [18]. Določili so pare genov, ki so pogosto izraženi skupaj, in naučili model, ki zna za vsak gen napovedati njegove pare. Kodirane gene so uporabili za napovedovanje interakcije med geni in dobili boljše rezultate od osnovnih modelov, s čimer so dokazali uspešnost modela gene2vec. Cheng in ostali so uporabili genske izraze za napovedovanje učinkovitosti porabe dušika za različne vrste koruze iz več vrst rastline Arabidopsis, ki je služila kot model [19]. Uspešno so pokazali, da dobijo boljše rezultate kot metode, ki so bile učene na samo eni rastlini. Chen in ostali so napovedovali odziv zdravil za raka na podlagi DNA [20]. Najprej so našli korelacije med genskimi izrazi in odzivi na zdravila in naučili model, ki iz DNA izlušči informativne vrednosti. Za uskladitev teh dveh modelov so uporabili učenje s prenosom in uspešno napovedali več kot 95 % celic, ki se odzovejo na določeno zdravilo.

II Predlagana metoda

Podatke smo dobili iz podatkovne baze Gene Expression Omnibus [21]. Naložili smo vse podatkovne nabore s človeškimi podatki in se osredotočili na tiste, ki so bili obdelani. Ker imajo podatkovni nabori različno število genskih izrazov (med 215 in 59.619), smo uporabili le izraze 884 genov, ki so bili odkriti kot pomembni z metodo L1000 [22]. Nazadnje smo pregledali vse podatkovne nabore in določili tiste, katerih vzorčne anotacije bi lahko uporabili kot ciljne spremenljivke. Na koncu smo imeli 70 podatkovnih naborov s skupno 6.713 primeri, 884 genskimi izrazi in 185 ciljnimi spremenljivkami. Podatkovne nabore smo razdelili na polovico v učno in testno skupino, povzetek uporabljenih podatkovnih naborov pa je v tabeli 3.1.

Za modele strojnega učenja smo uporabili dve različni arhitekturi nevronskih mrež: samokodirnike in večopravilne modele. Samokodirniki so nevronske mreže, narejene za stiskanje in rekonstruiranje vhodnih podatkov. Sestavljeni so iz dveh delov: kodirnika in dekodirnika. Obstajajo alternativne metode, ki ne uporabljajo nevronskih mrež za zmanjševanje dimenzij, kot so analiza glavnih komponent (PCA), singularni razcep in linearna diskriminantna analiza, vendar samokodirniki omogočajo nelinearno projekcijo podatkov. Glavni problem samokodirnikov je njihov neomejen latentni prostor, ki ga večopravilni model reši. Večopravilni modeli so nevronske mreže, ki iz enega vhoda napovejo več rezultatov hkrati. Tako je model prisiljen najti dobro projekcijo vhodnih podatkov glede na vsa opravila. Sestavljeni so iz dveh delov: kodirnika in ene glave za vsako opravilo. Enačbi za oba modela sta navedeni spodaj.

$$\begin{aligned} \tilde{x} &= samokodirnik(x) = dekodirnik\left(kodirnik(x)\right) \\ \tilde{y} &= ve\check{c}opravilni\_model(x) = glava\left(kodirnik(x)\right) \end{aligned} \tag{1}$$

Za učenje modelov smo primere iz učnih podatkovnih naborov razdelili na učne in validacijske primere v razmerju devet proti ena in jih naključno premešali. Za optimizacijo nevronskih mrež smo uporabili optimizator Adam [23], ki se zna izogibati lokalnim ekstremom in ima prilagodljive učne korake. Za funkciji izgube smo izbrali povprečno kvadratno napako (MSE) za samokodirnike in navzkrižno entropijsko izgubo za večopravilne modele (logloss). Za boljše razumevanje modelov smo izmerili tudi determinacijski koeficient (R^2) za samokodirnike in površino pod ROC krivuljo (AUC) za večopravilne modele. Za vsako arhitekturo smo naučili deset različnih modelov z različno velikimi kodirnimi plastmi. Za boljšo oceno stabilnosti smo vsak model učili desetkrat. Za lažje razumevanje bomo model z velikostjo kodirne plasti x imenovali x-model.

Za vsak naučen model smo evalvirali njegov kodirnik na vseh testnih podatkovnih naborih. Na kodiranih podatkih smo naučili model logistične regresije in uporabili križno validacijo za boljšo oceno AUC. Rezultate smo primerjali z modelom logistične regresije na nekodiranih podatkih, imenovanim osnova. Naredili smo AUC-AUC graf vseh testnih podatkovnih naborov, kjer so osnovni rezultati na x osi in rezultati učenega modela na y osi. Pri dobrem modelu bi točke ležale nad diagonalo grafa. Za lažje razumevanje smo za vsak model izračunali trendno črto. Izmerili smo tudi vrste napak, ki jih model naredi, in naredili prikaz stopnje resničnih pozitivnih primerov (TPR) glede na stopnjo resničnih negativnih primerov (TNR), TPR-TNR graf. Za boljšo predstavo o latentnih prostorih matrik smo vse testne podatkovne nabore zakodirali in naredili projekcijo prvih dveh komponent s PCA metodo.

III Eksperimentalna evaluacija

Pri učenju samokodirnikov je najboljše validacijske metrike dosegel 64-samokodirnik ($MSE = 135,1 \pm 3,3, R^2 = 0,983 \pm 0,001$), najslabše pa 4-samokodirnik ($MSE = 202,1 \pm 5,2, R^2 = 0,975 \pm 0,001$). Med večopravilnimi modeli je imel 16-večopravilni model najnižjo validacijsko navzkrižno entropijsko izgubo ($logloss = 0,406 \pm 0,008$), najvišji validacijski AUC pa 64-večopravilni model ($AUC = 0,816 \pm 0,010$). Najslabši je bil 4-večopravilni model ($logloss = 0,507 \pm 0,064, AUC = 0,670 \pm 0,060$). Opazili smo, da so večopravilni modeli z manjšo kodirno plastjo nestabilni, kar se je opazilo v dolgem učenju z minimalnimi spremembami in slabih končnih metrikah.

Pri testiranju modelov smo najprej pregledali AUC-AUC grafe. Pri obeh arhitekturah smo opazili, da so modeli z večjo kodirno plastjo boljši, kar potrjuje trendna črta, ki je pri teh modelih bližje diagonali. Opazili smo tudi, da obe arhitekturi izboljšata rezultate na podatkovnih naborih, kjer so bili prej rezultati slabši, in obratno. Najboljše rezultate je dosegel 64-večopravilni model ($y = 0,888x + 0,050$).

Na TPR-TNR grafih obeh arhitektur smo opazili, da večja kodirna plast vodi do višje TNR. Najvišjo TNR sta dosegla 64-samokodirnik in 64-večopravilni model ($TNR = 0,831 \pm 0,003$). Pri TPR pa se metrika viša samo do določene velikosti kodirne plasti, saj imajo modeli z največjo kodirno plastjo nižji TPR. Najvišjo TPR sta dosegla 16-samokodirnik ($TPR = 0,653 \pm 0,007$) in 32-večopravilni model ($TPR = 0,655 \pm 0,004$). Te vrednosti pomenijo, da večja velikost projektne plasti bolj pomaga pri klasifikaciji negativnih kot pozitivnih primerov.

V zadnjem delu testiranja smo analizirali kodirane primere s PCA grafi. Pri PCA projekciji nekodiranih primerov smo opazili jasno ločitev med podatkovnimi nabori. PCA projekcija kodiranih primerov iz samokodirnikov je pokazala podobno sliko, kjer se še vedno vidijo razlike med podatkovnimi nabori. Drugače je pri večopravilnih modelih, ki imajo dva različna primera projekcije, a oba zabrišeta meje med primeri različnih podatkovnih naborov. Samokodirniki tako ohranjajo strukturo podatkov, medtem ko večopravilni modeli vse primere kodirajo v isti prostor in ne razlikujejo med njimi.

IV Sklep

V tem magistrskem delu smo poskušali rešiti problem majhnih podatkovnih naborov z združevanjem takih naborov in učiti modele, ki bi smiselno kodirali genske izraze. Ovrednotili smo dve različni arhitekturi, samokodirnike in večopravilne modele. Samokodirniki so se izkazali za bolj stabilne pri učenju, večopravilni modeli pa so dosegali rahlo boljše rezultate med testiranjem. Kljub temu da so samokodirniki ohranjali strukturo podatkov, so večopravilni modeli bili bolj točni z drugo projekcijo. Obe arhitekturi pa sta pokazali, da je profil gensih izrazov možno predstaviti z nekaj informativnimi vrednostmi.

Chapter 1

Introduction

Understanding biological data is an integral part of biomedical research. We typically deal with large amounts of data, which often contains thousands of features and hundreds of samples from patients. Analyzing it by hand is impossible, which is where machine learning comes in. Machine learning in bioinformatics is used to analyze and extract results from data and can provide useful insights [1]. Researchers use it to identify candidate genes behind certain diseases [2], build protein interaction networks [3], or detect cancer cells in soft tissue scans [4]. There are many different types of biological data, from images to omics. In this thesis, we will focus on the omics.

Genomic data represents the fundamental biological data from which all other forms of data are derived. It refers to the structure and function of an organism's genome, which is defined as the complete set of instructions found in a cell, consisting of either deoxyribonucleic acid (DNA) or ribonucleic acid (RNA). A gene is defined as a section of DNA with a specific function, such as instructions for the synthesis of a particular protein. The DNA is transcribed into messenger RNA (mRNA), which is then translated by the ribosome to synthesise proteins, which are large molecules composed of amino acids. They perform and control a variety of functions in a cell, including metabolism, communication, the transport of small molecules, and protection [24]. Proteins have a direct influence on the phenotype of the cell.

The phenotype is an observable trait of a cell or an organism, which covers its physical appearance, development, biochemical properties, and behaviour. It is important to note that phenotype is not solely determined by genetics, rather it is influenced by the environment in which the organism resides [25].

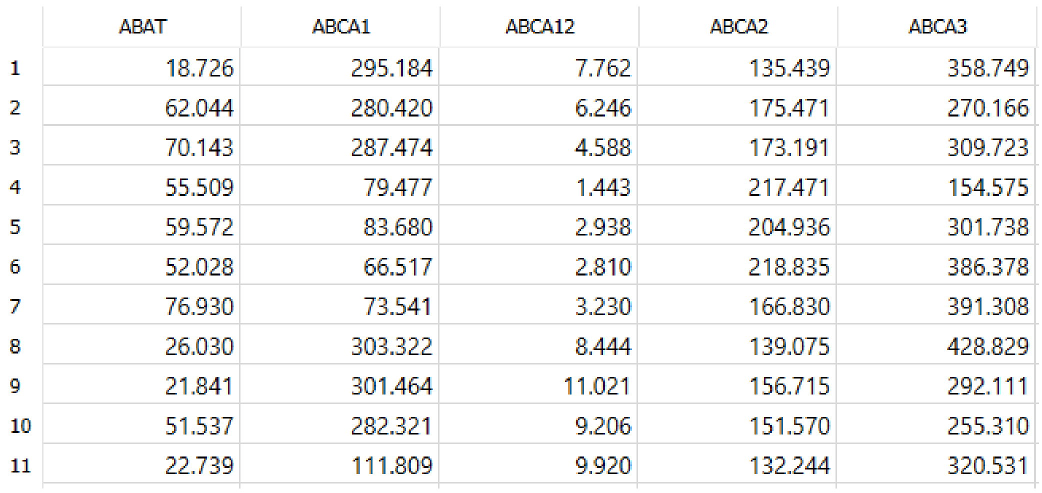
The above-described process is called gene expression and represents the direct connection between genotype and phenotype of an organism. This process can be regulated, primarily at the level of DNA transcription and to a lesser extent at the level of mRNA translation. Some genes regulate the expression of others and are referred to as promoters, enhancers, silencers, and insulators [5]. Additionally, the gene expression is altered with the age of the organism due to DNA damage, gender, and the time in the circadian rhythm of the organism [6]. The measurement of gene expression is vital for the identification of gene functions, the advancement of disease research, and the development of new drugs [7]. The ideal method for measuring gene expression would be to detect the number of proteins produced by the gene in question, which is a difficult process. Researchers have instead turned to indirect methods, such as measuring the number of mRNA strands and subsequently inferring the gene expression levels [26]. This can be achieved using methods such as microarray [27] and RNA sequencing [28]. An example gene expression data set is on Figure 1.1.

There are many public bioinformatics databases, including The Cancer Genome Atlas (TCGA)[1], which contains genetic data on different types of cancer, the Gene Expression Omnibus (GEO) [21], the ArrayExpress [8] and the Database of Genotypes and Phenotypes (dbGaP)[2], which contain gene expression and phenotype data, and UniProt [29], which contains protein data. These databases contain a huge amount of biological data, although a considerable number of the data sets have a limited number of samples.

The ArrayExpress database contains over 35,000 data sets with less than 10 samples. dbGaP has approximately 550 data sets with 10 samples or

[1]https://www.cancer.gov/tcga, accessed on 2.3.2024

[2]https://www.ncbi.nlm.nih.gov/gap/, accessed on 2.3.2024



	ABAT	ABCA1	ABCA12	ABCA2	ABCA3
1	18.726	295.184	7.762	135.439	358.749
2	62.044	280.420	6.246	175.471	270.166
3	70.143	287.474	4.588	173.191	309.723
4	55.509	79.477	1.443	217.471	154.575
5	59.572	83.680	2.938	204.936	301.738
6	52.028	66.517	2.810	218.835	386.378
7	76.930	73.541	3.230	166.830	391.308
8	26.030	303.322	8.444	139.075	428.829
9	21.841	301.464	11.021	156.715	292.111
10	51.537	282.321	9.206	151.570	255.310
11	22.739	111.809	9.920	132.244	320.531

Figure 1.1: A gene expression data set example. This is the GDS1112 data set from the GEO database. It is in matrix format, and we can see 11 samples with five genes: ABAT, ABCA1, ABCA12, ABCA2, and ABCA3. The raw gene expression values are the number of mRNA strands detected in each sample. These values are usually depth-normalised, as longer mRNA strands are easier to detect.

less, and GEO has approximately 1,900 such data sets. The distribution of data set sample size in those databases is presented in Figure 1.2. It is not uncommon for biological data sets to have small sample sizes, particularly when the data is costly and difficult to obtain. One such example is Hutchinson-Gilford progeria syndrome [9], an extremely rare genetic disorder that causes accelerated ageing. It is typically the consequence of a random mutation of the LMNA gene and is rarely hereditary. According to the Progeria Research Foundation[3], there are only 144 currently known cases of progeria in the entire world. The scarcity of these conditions makes it one of the most challenging genetic disorders to study. The use of deep machine learning tasks on such data sets is associated with higher uncertainty, which is referred to as the small data set problem [30].

The small data set problem has been addressed with many different methods. One of the more straightforward approaches is upsampling, where the

[3]https://www.progeriaresearch.org/quick-facts/, accessed on 3.3.2024

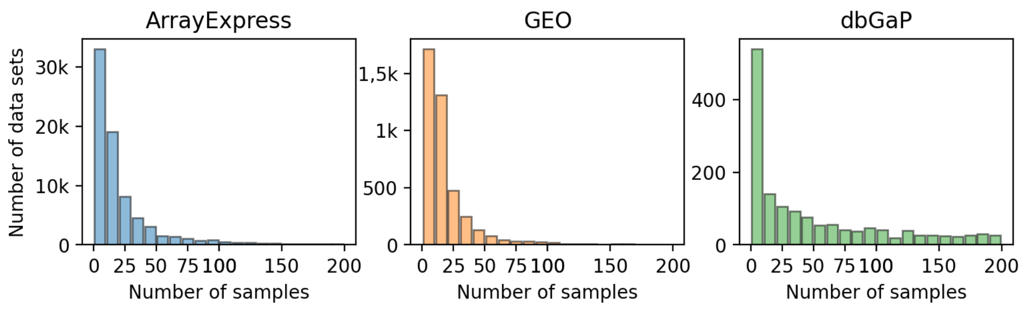


Figure 1.2: Data sets with few samples dominante in databases. The number of data sets is significantly reduced when the number of samples exceeds 10. Despite the relatively limited sample numbers, the largest data set in ArrayExpress has 27,887 samples, in GEO 202 samples, and in dbGaP 112,062 samples.

minority class samples are repeated to achieve a more balanced data set. Another method is SMOTE [31], which creates a new sample by interpolating between two close samples of the same class. A popular choice is the generation of synthetic samples with a generative model, such as variational autoencoders for protein variants [32] and diffusion models for generating proteins from gene expressions [33].

This thesis will focus on gene expression data and predicting phenotype traits in small data sets. We will address the small data set problem from a different perspective: by combining multiple small data sets into a larger one, a model can be trained that produces meaningful gene expression encodings for phenotype prediction. We hypothesise that by integrating the samples, the models will have a higher chance of learning a good latent representation and meaningfully embedding the data into the same concept space. The use of transfer learning to model smaller data sets could result in accurate and more intuitive models based on the encodings, as the knowledge will be pooled from many data sets with different tasks. We will investigate the effect of the embedding size on the final predictive performance, and how comparable are embeddings across different models.

The following section outlines the structure of the thesis. In Chapter 2 we provide an overview of the current gene expression databases, the evolution of phenotype prediction, and how gene expressions have been used in transfer learning. In Chapter 3 we describe the data gathering and selection process, the models that we used, and the training and testing regime. In Chapter 4 we present the training and testing results along with different embedding examples. In Chapter 5 we summarize our work and present suggestions for future improvements.

Chapter 2

Background

In this chapter, we present three gene expression databases in more detail, as they differ in size, purpose, and quality. Additionally, we present a short history of different techniques for phenotype prediction, and the current applications of gene expression in transfer learning for the prediction of observable organism traits.

2.1 Gene expression databases

There exist many publicly accessible databases for phenotype learning from gene expression data. We have previously referenced the ArrayExpress, the dbGaP, and the GEO database, which we will describe in greater detail here. All of the statistics presented here were gathered in May of 2024.

The ArrayExpress is a publicly accessible repository for functional genomics data. It was established in 2002 as an archive for freely accessible publication-related microarray data. It is now incorporated into the BioStudies [34] database, created in 2017 with the objective of unifying different biological modalities, such as epigenetics, RNA, and protein expressions. It has an R package called ArrayExpress[1] for programmatic access, which is part

[1]https://www.bioconductor.org/packages/release/bioc/html/ArrayExpress.html, accessed on 15.8.2024

of the Bioconductor project. The database contains data from studies that describe the functions and interactions between genes. Each data set is characterised by rich metadata attributes, such as sample annotation and data collection protocol, and includes raw and processed data. It currently contains data from 77,983 studies, 3,445 different species and approximately 21 million samples. Approximately half of the studies included in the database contain data sets with 10 samples or less.

The dbGaP is a database for research that examines the genotype-phenotype interaction. This includes genome-wide association studies, medical sequencing, molecular diagnostic assays, and associations between genotype and non-clinical traits. Approximately half of the data sets are publicly accessible, while the remainder require authorization. The database stores phenotypes separately from genotypes. Phenotype data is stored in matrix format, and genotype data is stored in either individual files or one of three formats: matrix, VCF [35], or PLINK [36]. The database contains 14,403 data sets, of which 2,983 are gene expression data sets with 390,846 samples. In contrast to ArrayExpress, only a fifth of all gene expression data sets contain 10 samples or less.

The GEO database is an international public repository that contains high-throughput functional genomic data. It was established in 2002 as a resource for gene expression studies, where users publish the data themselves as a series of objects, representing a group of samples from the same experiment. The moderators then curate the data into a collection of biologically and statistically comparable samples that form a data set, ensuring the optimal quality of published data. The GEO database can be accessed programmatically through a tool called E-Utils[2] for enhanced search and retrieval. The database contains 4,348 data sets with 7,014,414 samples from 15 different species. Each data set is accompanied by sample annotations, with the number of labels per sample ranging from 2 to 100. Of these data sets, 1,994 contain 10 samples or fewer.

[2]https://www.ncbi.nlm.nih.gov/geo/info/geo_paccess.html, accessed on 15.8.2024

2.2 Phenotype prediction

Phenotype prediction is the process of predicting observable characteristics of an organism, whether external or internal. The challenge lies in the complex interplay between genotype and environmental factors, which exert a significant influence on the phenotype. A deeper understanding of phenotypes can lead to identifying genetic risk factors for diseases, thus enabling early diagnosis and the implementation of personalised treatment plans. A search of the PubMed database[3] using the keywords "gene expression phenotype" returned approximately 200,000 research articles, indicating that the research field of gene expression and phenotype interaction is indeed active, with 40,000 papers published in the last four years alone. However, when we add the keyword "prediction", the number of research articles returned is reduced to 17,000. This is because a significant proportion of gene expression and phenotype research is focused on identifying interactions and not prediction. The following section will present some of the early examples of papers in the field of gene expression data related to phenotype prediction.

In 2002, Beer et al. [10] realised that microscopic examination of tissue was insufficient for prediction of disease progression and clinical outcome in patients with lung cancer. They proposed that gene-expression profiles derived from microarray data could be used to more accurately predict patient survival in the early stages of the disease and identify high-risk patients. They removed the genes with the lowest expression levels and performed hierarchical clustering, which confirmed high associations between the cluster and the stage of the tumour. They identified a total of 967 genes as significantly different between early and late-stage tissues, thereby proving that gene expression could be used for prediction. They created a 50-gene risk index, a set of 50 genes that could differentiate between low- and high-risk patients. They confirmed their findings by testing the method on an independent data set, and obtained similar results.

[3]https://pubmed.ncbi.nlm.nih.gov/, accessed on 15.8.2024

Another example is the study conducted by Baty et al. in 2006 [11] that analysed the potential for gene expression manipulation, and examined how different beverages influenced the peripheral blood gene expressions. The researchers tested their hypothesis by providing the participants with different beverages (grape juice, red wine, diluted ethanol, and water) and regularly taking blood samples throughout the next 12 hours, from which they extracted gene expressions. They used linear regression to model the expressions at different timestamps and demonstrated that all four beverages influenced gene expressions differently, but this was only evident when the authors used instrumental variables analyses instead of between-group analyses.

In 2012, Schramm et al. [12] used gene and exon expressions to predict patient outcomes in individuals with neuroblastoma, a type of cancer affecting nerve tissue. The researchers focused their analysis on tumour-level gene and exon expressions, which are protein-encoding gene regions. During the data processing stage, they removed all features that they deemed to be of no use and performed an F-test to check for expression variance equality between classes. They retained only those that showed significant association with the class label across multiple test repetitions. They used a method known as shrunken centroids [37], which is based on the nearest centroid approach, to predict relapse and overall survival. They achieved the prediction accuracy of 83.6% and tested the robustness of their predictors with backward logistic regression. They confirmed that gene and exon expressions were the only significant predictors of overall survival prediction among the other variates.

In 2020, Part et al. [13] used deep neural networks to diagnose Alzheimer's disease. The authors used three data sets from the GEO database, with emphasis on the prefrontal cortex: two with gene expression values and one with methylation profiles. Given that the authors were working with different data types, they adopted a strategy of feature selection based on differentially expressed genes and differentially methylated positions, utilising a t-test. Common features were then extracted by performing an intersection.

They used Bayesian optimization to determine the optimal hyperparameters of a deep neural network, which was then trained. The authors tested their model against multiple baseline models and demonstrated that it outperformed them by a large margin.

2.3 Gene expression transfer learning

Transfer learning [14] is a machine learning problem that involves acquiring knowledge in one domain and applying it to another. It is a common practice to utilise pre-trained models as a starting point, particularly in the domains of image classification [15] and natural language processing [16], where there are vast amounts of data. A search on the PubMed website for the previously mentioned keywords in conjunction with ”transfer learning” yielded fewer than 1,000 results, which suggests that this research field is still in its infancy. The majority of studies in this area focus on demonstrating the effectiveness and utility of a learned method when applied to a different organism.

Choi et al. [17] researched the potential of neural networks to provide a vector embedding on gene expression data sets through the process of gene co-occurrence. They used data from 36 cancer types in the TCGA database and removed samples they deemed to be of poor quality. They trained a shallow neural network to create embeddings for gene expressions and samples, analogous to a matrix factorization approach. They showed that the model learned relationships between samples, clustering the same cancer types together using self-organising maps [38]. To demonstrate the usefulness of the produced embeddings they classified molecular subtypes of liver cancer, replicating the same clusters as the original study.

Du et al. [18] developed a method, designated gene2vec, which is based on word2vec [39] for gene embeddings. The authors utilised all available human gene data sets from the GEO database, quantile-normalised them, and used Pearson’s correlation to determine gene co-expression pairs from each data set. Subsequently, they used gene ontology annotations to define gene pairs with similar annotations as positive examples and the remainder as negative. Ultimately, they encoded each gene using the one-hot encoding method. They trained a simple, one-layer neural network to predict a co-expressive gene from the other in a pair, with the hidden layer output serving as a gene embedding. When they visualised the embeddings, they observed that tissue-specific genes were clustered. They further tested the usefulness of encodings by training a separate model for the gene-gene interaction task, which resulted in an AUC score of 0.72, better than the baseline of 0.66. This demonstrates that the encodings of the gene2vec model were effective, as the only information used for testing were the names of the genes.

Cheng et al. [19] predicted nitrogen use efficiency across corn varieties from Arabidopsis, which served as a model organism. The researchers gathered gene expression data from the plants they had cultivated and performed a feature selection to reduce the number of genes. They trained multiple tree-based models with boosting to predict nitrogen use efficiency and used feature importance from the trained models and gene regulatory networks to validate their results. Their approach to finding important genes across species demonstrated superior performance compared to other methods, where only one species was present.

Chen et al. [20] employed transfer learning to predict cancer drug responses to RNA sequences with their model called scDEAL. The researchers used denoising autoencoders to learn a compact representation of gene expressions in specific cells and to learn the correlation between drug responses and gene expressions. They trained an additional autoencoder to extract features from single-cell RNA sequences, and used transfer learning to train another deep model to adapt the extracted gene features to have similar distributions. The final prediction was made by taking a single-cell RNA sequence along with the adapted features and returning whether a certain cell was sensitive or resistant to a particular drug. The researchers correctly predicted 97.1% of drug-resistant and 95.8% of drug-sensitive cells.

In their study, Hanczar et al. [40] used transfer learning to predict different types of cancer from two distinct data sources: gene expressions, compiled by Torrente et al. [41], and RNA sequences from the TCGA. The authors used both supervised and unsupervised learning techniques. They focused on evaluating the performance of deep neural networks based on model hyperparameters, which they discovered to exert minimal influence on the final results. They compared their deep model to other machine learning methods exhibited superior performance for large data sets, and demonstrated that for small data sets transfer learning can strongly improve the model performance.

Chapter 3

Methodology

In this chapter, we outline our data selection process and the data sets we ended up using. We present the two deep neural network architectures that we utilized, along with their associated loss functions. Finally, we describe our training, testing and evaluating procedure, as well as the experimental setup with all of the parameters used.

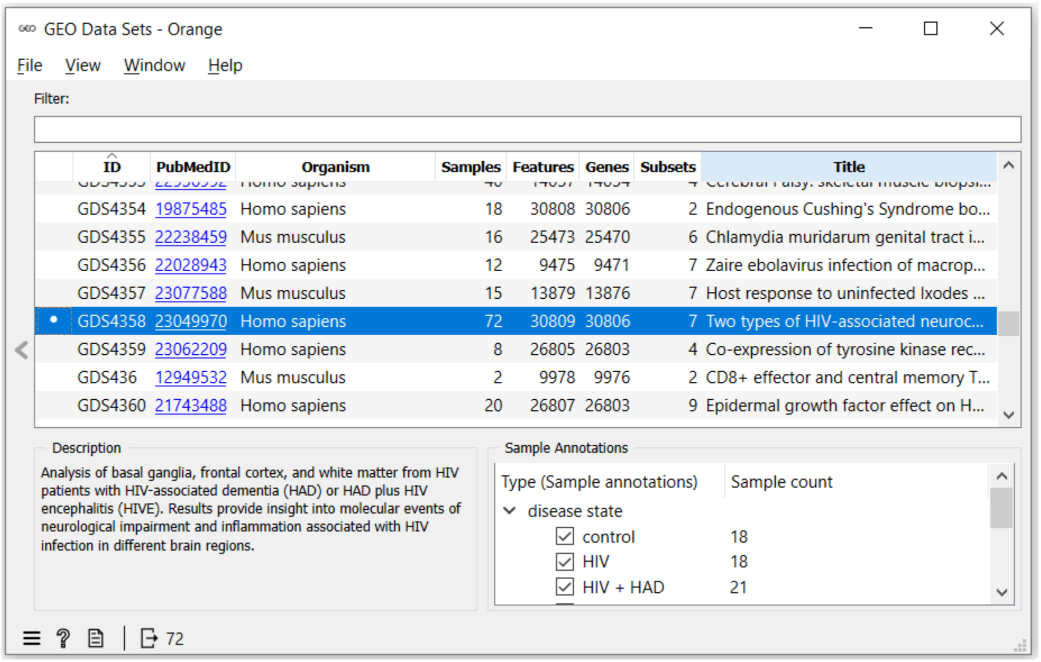
3.1 Data selection

We decided to use the GEO database [21], an international public repository containing microarray, next-generation sequencing, and other forms of genomics data submitted by the community. We accessed the GEO database through the data mining program Orange [42] and its Python libraries serverfiles[1] and orange3-bioinformatics[2]. An illustrative example of a GEO data set is provided in Figure 3.1.

We have selected data sets representing human data whose values have already been processed as there is a significant value variability in data sets with raw values. Furthermore, there is considerable variation in the number of gene expressions across the data sets. The smallest data set is GDS995

[1]https://github.com/biolab/serverfiles/, accessed on 24.11.2022

[2]https://github.com/biolab/orange3-bioinformatics/, accessed on 24.11.2022



GEO Data Sets - Orange

File View Window Help

Filter:

	ID	PubMedID	Organism	Samples	Features	Genes	Subsets	Title
	GDS4354	19875485	Homo sapiens	18	30808	30806	2	Endogenous Cushing's Syndrome bo...
	GDS4355	22238459	Mus musculus	16	25473	25470	6	Chlamydia muridarum genital tract i...
	GDS4356	22028943	Homo sapiens	12	9475	9471	7	Zaire ebolavirus infection of macrop...
	GDS4357	23077588	Mus musculus	15	13879	13876	7	Host response to uninfected Ixodes ...
•	GDS4358	23049970	Homo sapiens	72	30809	30806	7	Two types of HIV-associated neuroc...
	GDS4359	23062209	Homo sapiens	8	26805	26803	4	Co-expression of tyrosine kinase rec...
	GDS436	12949532	Mus musculus	2	9978	9976	2	CD8+ effector and central memory T...
	GDS4360	21743488	Homo sapiens	20	26807	26803	9	Epidermal growth factor effect on H...

Description

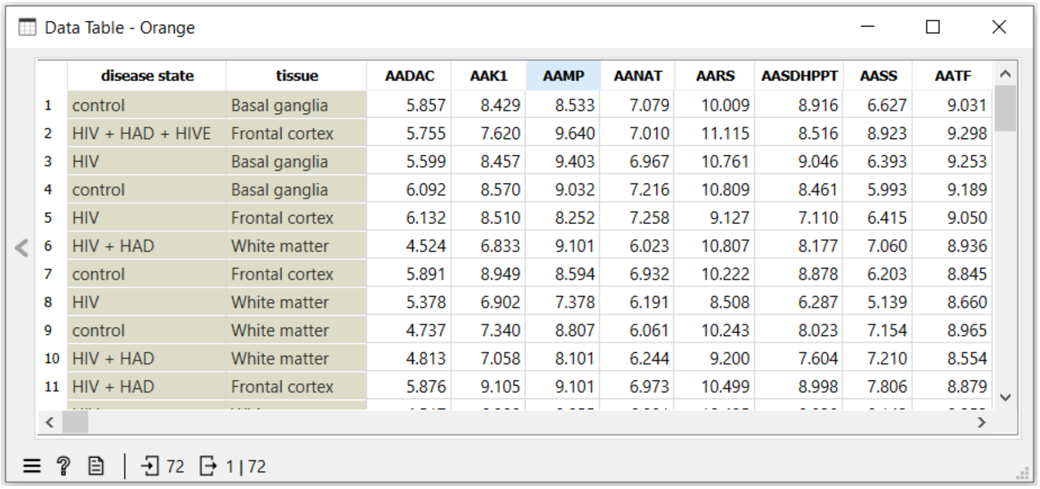
Analysis of basal ganglia, frontal cortex, and white matter from HIV patients with HIV-associated dementia (HAD) or HAD plus HIV encephalitis (HIVE). Results provide insight into molecular events of neurological impairment and inflammation associated with HIV infection in different brain regions.

Sample Annotations

Type (Sample annotations)	Sample count
disease state	
✓ control	18
✓ HIV	18
✓ HIV + HAD	21

72

(a) An overview of different GEO data sets with their statistical summary. Data set GDS4358 is selected, for which a description and sample annotations are displayed.



Data Table - Orange

	disease state	tissue	AADAC	AAK1	AAMP	AANAT	AARS	AASDHPPT	AASS	AATF
1	control	Basal ganglia	5.857	8.429	8.533	7.079	10.009	8.916	6.627	9.031
2	HIV + HAD + HIVE	Frontal cortex	5.755	7.620	9.640	7.010	11.115	8.516	8.923	9.298
3	HIV	Basal ganglia	5.599	8.457	9.403	6.967	10.761	9.046	6.393	9.253
4	control	Basal ganglia	6.092	8.570	9.032	7.216	10.809	8.461	5.993	9.189
5	HIV	Frontal cortex	6.132	8.510	8.252	7.258	9.127	7.110	6.415	9.050
6	HIV + HAD	White matter	4.524	6.833	9.101	6.023	10.807	8.177	7.060	8.936
7	control	Frontal cortex	5.891	8.949	8.594	6.932	10.222	8.878	6.203	8.845
8	HIV	White matter	5.378	6.902	7.378	6.191	8.508	6.287	5.139	8.660
9	control	White matter	4.737	7.340	8.807	6.061	10.243	8.023	7.154	8.965
10	HIV + HAD	White matter	4.813	7.058	8.101	6.244	9.200	7.604	7.210	8.554
11	HIV + HAD	Frontal cortex	5.876	9.105	9.101	6.973	10.499	8.998	7.806	8.879

72 1 | 72

(b) An example of gene expression values for data set GDS4358, which has two sample annotations, 'disease state' with four values and 'tissue' with three values.

Figure 3.1: An example of a GEO gene expression dataset. The figures illustrate a screenshot from the Orange program depicting a selection of a GEO data sets (upper) and a view of data set GDS4358, which contains data about HIV-associated neurocognitive impairment from two brain regions (lower).

(PubMed ID 15506941) with 215 gene expressions. In contrast, the largest data set is GDS1798 (PubMed ID 15834008) with 59,619 gene expressions. We have decided to retaingene expressions for genes identified as significant by a profiling method L1000 [22]. There, the authors have identified 978 landmark genes suitable for inferring the expression levels of 81% of all other genes. By keeping only these genes, we greatly reduce the size of the data while still retaining a substantial amount of information. To ensure consistency across data sets, we further identified the most common gene expression variables and only kept the data sets that had them for further analysis.

We used principal components analysis (PCA) [43] to further check the compatibility of gene expression values between data sets and excluded those that significantly deviated from the rest. We then inspected each data set and determined whether any of its sample annotations could be used as target variables for prediction purposes. We combined the sample annotations with multiple labels into two groups and split them into multiple one-versus-rest sample annotations when necessary. Many sample annotations contained metadata, such as sample identification numbers and the age of a sample, which were discarded. We kept those describing tissue type, genetic variation, and disease state. Based on the results of our initial experiments, we decided to keep only data sets with 50 or more samples.

Ultimately, we selected 70 data sets with a total of 6,713 samples, 884 gene expression variables, and 185 sample annotations for prediction. For each annotation, we ensured that the target class was in the minority. Furthermore, we split the data sets into two groups for training and testing purposes, with each group consisting of 35 data sets and a similar number of samples. The train group has 3,334 samples and the test group has 3,379 samples. A summary of all data sets can be found in Table 3.1.

Name	N	A
GDS4404	50	2
GDS3829	50	3
GDS5074	52	1
GDS4167	52	1
GDS4299	52	1
GDS4513	53	1
GDS4906	54	3
GDS4896	54	3
GDS4412	56	3
GDS2643	56	5
GDS3459	56	4
GDS5093	56	3
GDS4266	58	3
GDS3627	58	1
GDS4607	60	2
GDS4056	61	5
GDS4379	62	5
GDS4176	62	3
GDS3057	64	3
GDS5083	64	1
GDS4381	64	1
GDS4587	66	1
GDS4198	70	3
GDS5205	70	1
GDS4358	72	6
GDS4471	76	5
GDS4282	76	5
GDS4103	78	1
GDS4758	79	3
GDS3329	79	1
GDS4181	80	1
GDS4975	81	1
GDS3539	82	4
GDS5277	86	3
GDS4088	86	1
GDS4837	88	2
GDS4336	90	2
GDS4761	91	7
GDS3885	92	3
GDS4456	93	1
GDS4182	96	1
GDS4562	96	2
GDS4968	99	1
GDS4273	103	2
GDS4057	103	5
GDS4130	104	2
GDS4516	104	5
GDS3257	107	4
GDS4318	108	3
GDS2767	108	4
GDS5037	108	4
GDS4549	116	3
GDS4129	120	1
GDS3837	120	1
GDS5393	120	3
GDS4222	130	5
GDS4274	130	1
GDS5000	131	2
GDS5363	139	2
GDS5499	140	3
GDS4267	154	3
GDS4278	154	1
GDS5027	156	3
GDS3952	162	3
GDS3312	163	1
GDS4600	170	1
GDS4296	174	9
GDS4602	180	1
GDS2771	192	1
GDS4206	197	3

Table 3.1: Summary of used GEO data sets. For each data set we display its GEO name 'Name' as well as the number of samples 'N' and the number of sample annotations 'A'. The data sets in train group are presented in **bold**, while the rest belong to the test group.

3.2 Neural networks

A neural network is a computational model of machine learning, designed to imitate the functionality of the neurons in a human brain. It is composed of multiple neurons, arranged in layers with activation functions in between. A basic neural network has two layers: a hidden layer and an output layer. In contrast, a deep neural network has multiple hidden layers. Each neuron in a layer performs a weighted sum of its inputs, adds bias, and applies a non-linear activation function. This nonlinearity ensures that the model is capable of learning non-linear relations, otherwise, a neural network would only be able to solve linear problems. The equation for a neuron i in a layer l is therefore

$$\begin{aligned} z_i^l &= \sum_j w_{ij}^l a_j^{l-1} + b_i^l \\ a_i^l &= \sigma\left(z_i^l\right) \end{aligned} \tag{3.1}$$

where a_j^{l-1} is the activation value of the neuron j in a previous layer $l-1$, w_{ij}^l is the weight of the connection between neurons j and i, and b_i^l is the bias of the neuron i. The activation function, denoted by $\sigma(\cdot)$, is a non-linear function. Commonly used activation functions are ReLU $\sigma(x) = max(0, x)$ and sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. The activation value of the neuron, a_i^l, represents the output of the neural network in case where the neuron is in the final layer. The weights and biases are most often tuned through the process known as backpropagation [44]. This involves calculating the derivatives of the weights and biases, followed by gradient descent to improve the overall performance of a neural network and decrease its loss function.

3.2.1 Autoencoders

Autoencoders [45] are a class of neural networks designed to compress and reconstruct unlabeled data, thereby reducing the dimensionality of the input data. Many similar approaches do not employ the use of neural networks for this purpose, such as PCA, singular value decomposition (SVD) [46], and linear discriminant analysis (LDA) [47]. Unlike these methods, autoencoders allow for non-linear data projection.

The concept of autoencoders was first introduced by Kramer in 1991 [48] as a non-linear generalisation to PCA, called autoassociators. They were initially used primarily for dimensionality reduction. However, they have since been utilised for image denoising [49], anomaly detection [50], and language translation [51].

An autoencoder consist of two neural networks: an encoder and a decoder. The encoder is responsible for learning how to encode the data and project it into a lower-dimensional subspace. The decoder learns how to reconstruct the data given the encoding. The equations for an encoder and a decoder are

$$\begin{aligned} z &= encoder(x) \\ \tilde{x} &= decoder(z) \end{aligned} \tag{3.2}$$

where x is the input, z represents the encoded input, which is of lower dimensionality than x, and $\tilde{x}$ is the decoded output or reconstructed input. An illustration of the architecture of an autoencoder is in Figure 3.2.

Autoencoders do not require labels during training, which allows them to be trained on unlabeled data and be used for pretraining to learn general features and concepts. Following training, the decoder is typically discarded, and the encoder becomes a part of a bigger network where it is further trained on a specific task.

One limitation of autoencoders is that the latent space Z is not constraint and can assume any value. A minor alteration of the input typically results in a completely different encoding, which can cause problems for further tasks as similar inputs are no longer similar. This issue can be addressed through different approaches. One potential solution is to enforce a structure of the latent space by introducing an additional loss function, such as an absolute loss or a square loss. In a variational autoencoder, a Kullback–Leibler divergence loss is used to guarantee that the latent space is distributed nor-

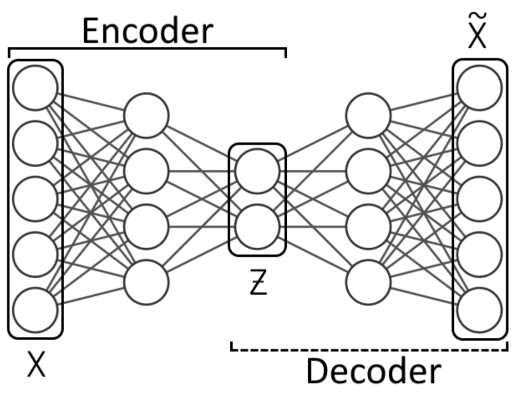


Figure 3.2: Autoencoder architecture example. The leftmost layer represents the input x, the middle layer the encoding z, and the rightmost layer is the reconstructed input $\tilde{x}$. The first three layers are part of the encoder, while the last three are part of the decoder. It is typical of encoders and decoders to have symmetrical architectures.

mally. This additional loss must be balanced with the overall network loss and introduces another hyperparameter, which may be challenging to set. An alternative approach is to make the latent space aware of the task it is intended to solve with a a multi-task model.

3.2.2 Multi-task models

Initially, neural networks were designed for a single predictive task only, and transfer learning has been employed to reuse models for new tasks. Researchers have proposed the concept of universal feature extractors, analogous to autoencoders, that can simultaneously learn several different tasks. Such models are designated as multi-task models. Bilen and Vedaldi [52] proposed such a model where all parameters across tasks are shared, with the exception of those in the normalization layers. Rebuffi et al. [53] proposed a model where the majority of the parameters are shared between tasks.

A multi-task model consists of two components: a backbone and a prediction head. The backbone has the same function as an encoder in an autoencoder model, that is to provide an encoding of an input. The pre-

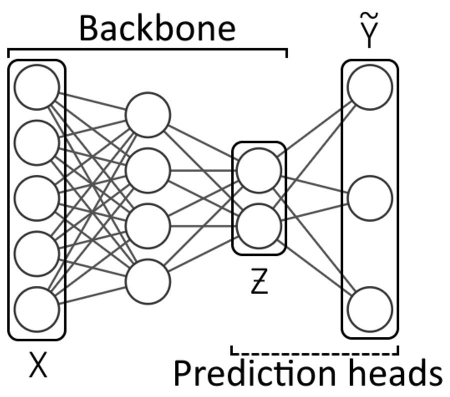


Figure 3.3: Multi-task architecture example. The leftmost layer represents the input x, and the third layer is the encoding z. The first three layers are part of the backbone of the model. The final layer consists of three neurons or prediction heads and makes the final prediction $\tilde{y}$. Each sample is associated with at least one prediction head, with multiple heads for multi-label tasks.

diction head takes that encoding and makes a prediction based on it. Each task has its own prediction head, and thus a multi-task model has multiple prediction heads. The equations for the multi-task model are

$$\begin{aligned} z &= backbone(x) \\ \tilde{y} &= head_j(z) \end{aligned} \tag{3.3}$$

where x represents the input, while z represents the encoded input and output of the backbone. z is then processed by a prediction head $head_j(\cdot)$ belonging to the j-th task and produces a prediction $\tilde{y}$. The architecture of each prediction head can range from a single layer to a more complex deep model. An illustration of the architecture of a multi-task model is in Figure 3.3.

The backbone of a multi-domain model performs a similar function to that of an encoder in an autoencoder model, but it differs in one detail. The encodings of an autoencoder are designed with input reconstruction in mind and are not designed to predict any specific task. In contrast, the backbone in the multi-task model is designed with task prediction in mind and is trained so that the encodings are constructed for task prediction purposes.

3.3 Training procedure

The samples from the training data sets were divided into two distinct groups: the training data and the validation data. The training data comprised 90% of the samples, while the validation data comprised the remaining 10%. We divided the samples from each data set separately so that each prediction head from the multi-task model had access to both the training and validation data. We then generated three vectors for each sample: x represented the gene expression data, y represented the target classes for each task of the sample, and w is a binary vector that tracks the tasks to which this sample belongs. Vectors y and w have the same dimensions. This enables the loss of a multi-task model to be masked with w and consider only the predictions from the appropriate heads for each sample. We then randomly shuffled the training data to mix samples from multiple data sets together, as it ensures that different samples are placed in the same batch and facilitates more stable training.

We used the Adam optimizer [23] for parameter updates, which utilises gradient momentum and adaptive learning steps to avoid local minima and perform better parameter updates. During the training of the model we used early stopping with learning rate scheduling. When the early stopping was activated, we reduced the learning rate of the Adam optimizer and resumed the training from the most recent best model.

We used two different loss functions, one for the autoencoders and one for the multi-task models. The loss function for the autoencoders was the mean squared error (MSE), also known as ridge or L2 loss, which is a common regression loss. MSE measures the average squared difference between an input and the corresponding output. The loss function for multi-task models was cross-entropy loss, also known as log loss. It is defined as the logarithm of the likelihood function and measures the closeness of the predicted probability to the true class. The formulas for MSE and log loss are given in Equation 3.4 and Equation 3.5 respectively.

$$mse(X, \tilde{X}) = \frac{1}{n} \sum_{x, \tilde{x} \in X, \tilde{X}} (x - \tilde{x})^2 \tag{3.4}$$

$$logloss(Y, \tilde{Y}) = -\frac{1}{n} \sum_{y, \tilde{y} \in Y, \tilde{Y}} ylog(\tilde{y}) + (1 - y)log(1 - \tilde{y}) \tag{3.5}$$

For each model, we calculated an alternative metric to better understand its performance. In the case of autoencoders, we calculated the coefficient of determination (R^2), which provides an estimate of the proportion of variance explained by the model. The formula is given in Equation 3.6, where SS_{res} represents the sum of squares of residuals and SS_{tot} represents the total sum of squares. The mean regressor achieves $R^2 = 0$, whereas a regressior with perfect output achieves $R^2 = 1$. In the case of multi-task models, we used the area under the receiver operating characteristic curve (AUC-ROC or AUC), which is a classification measure of how well the model can distinguish between positive and negative samples. This is calculated as the probability of a random positive sample having a higher predicted probability than a random negative sample. A random classifier achieves $AUC = 0.5$, while a perfect classifier achieves $AUC = 1$.

$$\begin{gathered} R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \\ SS_{res} = \sum_{x, \tilde{x}} (x - \tilde{x})^2, \; SS_{tot} = \sum_{x} (x - \bar{x})^2 \end{gathered} \tag{3.6}$$

3.4 Testing procedure

Following the training of the model, we evaluated its encoder for prediction purposes on the testing data sets. We calculated the encoding for each sample in a data set, then trained a logistic regression model with those encodings as inputs. We used Leave-one-out cross-validation (LOOCV) to estimate the performance of the logistic regression more accurately. We calculated log loss, accuracy, precision, recall, AUC, and confusion matrix for each data set, but focused predominantly on AUC. For each data set, we measured the performance of logistic regression on unencoded data as a baseline.

To evaluate the performance of a given model, we constructed an AUC-AUC plot, which plots the baseline AUC on the x-axis against the AUC of a logistic regression on encoded data on the y-axis for each testing data set. In order for the model to be better than the baseline, it is necessary to demonstrate an improvement in the results obtained on individual testing data sets. This is indicated by the AUC-AUC points on the plot lying above the diagonal line ($y \geq x$). To obtain a more accurate estimation of the performance of the model against the baseline, we calculated a trend line from the AUC-AUC points. A trend line for a baseline model would have a slope coefficient equal to 1 and an intercept set to 0. Due to previous experiments, we expect that our models will have an intercept greater than 0 and a slope less than 1, which would signify improved performance for lower-performing data sets and degraded performance for higher-performing data sets.

We constructed a confusion matrix for each model by combining the confusion matrices from individual test data sets. This was done in order to ascertain the type of errors most commonly made by our models. Given that the confusion matrix has four dimensions, we decided to plot the true positive rate (TPR) against the true negative rate (TNR) in a TPR-TNR plot. The formulas for TPR and TNR are provided in Equation 3.7 and Equation 3.8 respectively, where TP denotes all correctly predicted positive samples, T all positive samples, TN all correctly predicted negative samples, and N all negative samples.

$$TPR = \frac{TP}{P} \tag{3.7}$$

$$TNR = \frac{TN}{N} \tag{3.8}$$

In order to gain a deeper insight into the encodings produced by the models, we analysed them with PCA and plotted the first two dimensions, which are the most informative. Additionally, we calculated the explained variation of each projected dimension in order to gain a better understanding of the data structure and determine the number of dimensions required to sufficiently account for the observed variation.

3.5 Experimential setup

We used the programming language Python version 3.10.6[3] to conduct the practical experiments. We used the PyTorch library [54] for the implementation of autoencoder and multi-task model architectures, loss functions and Adam optimiser, the TorchMetrics library [55] for R^2 and AUC calculations, and the scikit-learn library [56] for logistic regression, LOOCV, and all other test metrics. We used the numpy library [57] for data manipulation and the pandas library [58] for data storage purposes. We trained the models on an NVIDIA GeForce GTX TITAN X graphics card and tested them on an Intel Xeon E5-2650 v4 processing unit. We selected all values manually based on the results of our preliminary experiments.

We trained all models for 1,000 epochs with a batch size of 64. The initial training phase laster for 100 epochs, after which we initiated early stopping with a patience period of 25 epochs. Once early stopping had been activated, we reduced the learning rate by a factor of 10. We terminated training after three reductions of the learning rate. Early stopping was activated when the loss function demonstrated no improvement over the duration of a patience period, which was defined as a change of 1 for MSE or 10^{-3} for log loss. For the Adam optimizer we used a learning rate of $\eta = 5 \cdot 10^{-5}$, and the default momentum coefficients of $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We trained each model 10 times for uncertainty estimation, with the validation portion of the training data randomly selected each time.

We used a logistic regression model with an L2 penalty for testing. We set the regularization strength to $\lambda = 0.01$ and rebalanced the weight of classes

[3]https://www.python.org/downloads/release/python-3106/, accessed on 28.8.2022

to ensure equal importance. We used the LGBFS solver with a maximum number of iterations set to 10,000 to guarantee convergence.

In regard to the model architecture, we opted for simple and shallow networks. As demonstrated by Hanczar et al. [40], the number of layers in the backbone architecture is of minimal importance, and our initial experiments indicated minor differences in performance between layer sizes. Consequently, we used three layers in encoders, with the first two having a size of 512 neurons each. We varied the final latent layer in size from 4 to 64, which represented the size of the encodings. In their study, Choi et al. [17] employed encodings of size 50, which was more than sufficient for their gene expression embedding analysis. Therefore, we chose the encodings in the same number range. All layers were equipped with a ReLU activation function. For the autoencoders, we simply mirrored the architecture of the encoder to create the decoder. For multi-task models, we added a linear layer with a sigmoid activation function, which acted as a layer of prediction heads.

Chapter 4

Results

In the first part of this chapter, we describe the training results of both architectures. In the second part of this chapter, we present the performance of our models on test data sets when compared to a baseline, and we examine how the structure of the data changes when we encode it with our models. For the sake of simplicity, we will refer to a model with the latent size of x as an x-model. Thus, the autoencoder with a latent size of 8 will be referred to as an 8-autoencoder, and the multi-task model with a latent size of 32 will be referred to as a 32-multitask model.

4.1 Training

In order to plot the results of all models with the same latent size, we averaged and calculated the standard deviation of the metrics of the models that were still trained at certain epochs. As a result the uncertainty for the epochs of the model that trained the longest is not available. However, the earlier epochs provide a representative picture of how a model behaves during training. Further detailed results on training are available in Appendix A.

Figure 4.1 shows the progression of autoencoder metrics during the training process. All models demonstrated stable training with minimal differences between the models of the same latent size. All models achieved com-

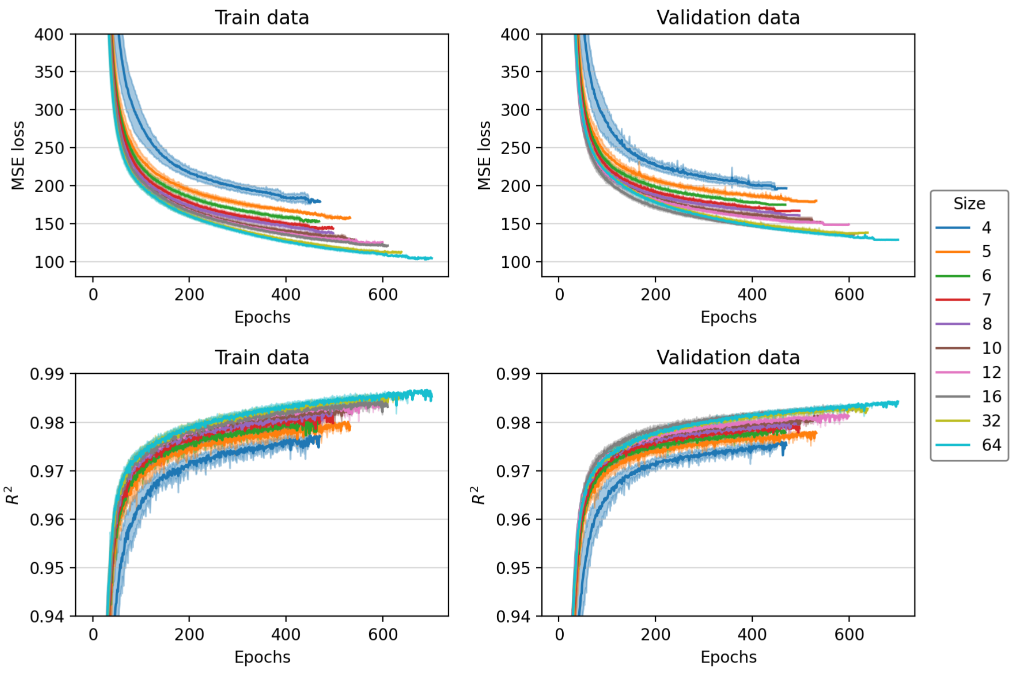


Figure 4.1: Autoencoder architecture exhibits stable training. The upper two figures present the MSE loss on train and validation data for the autoencoder models, while the lower two figures present the R^2 score on train and validation data. The color in the legend represents the size of the latent layer of the encoder of the model. The 64-autoencoder achieved lowest MSE loss and highest R^2 score for both train and validation data.

parable results on training and validation data, which demonstrates that the models did not overfit. The model with the best validation MSE and R^2 scores was the 64-autoencoder ($MSE = 135.1 \pm 3.3, R^2 = 0.983 \pm 0.001$), while the model with the poorest validation metrics was the 4-autoencoder ($MSE = 202.1 \pm 5.2, R^2 = 0.975 \pm 0.001$).

Figure 4.2 presents the progression of the log loss during the training process of multi-task models. As we found that the training of multi-task models was highly unstable for models with smaller latent sizes, we have plotted the metrics separately for unstable models (4-, 5-, 6-, and 7-multitask model) and stable models (8-, 10-, 12-, 16-, 32-, and 64-multitask model).

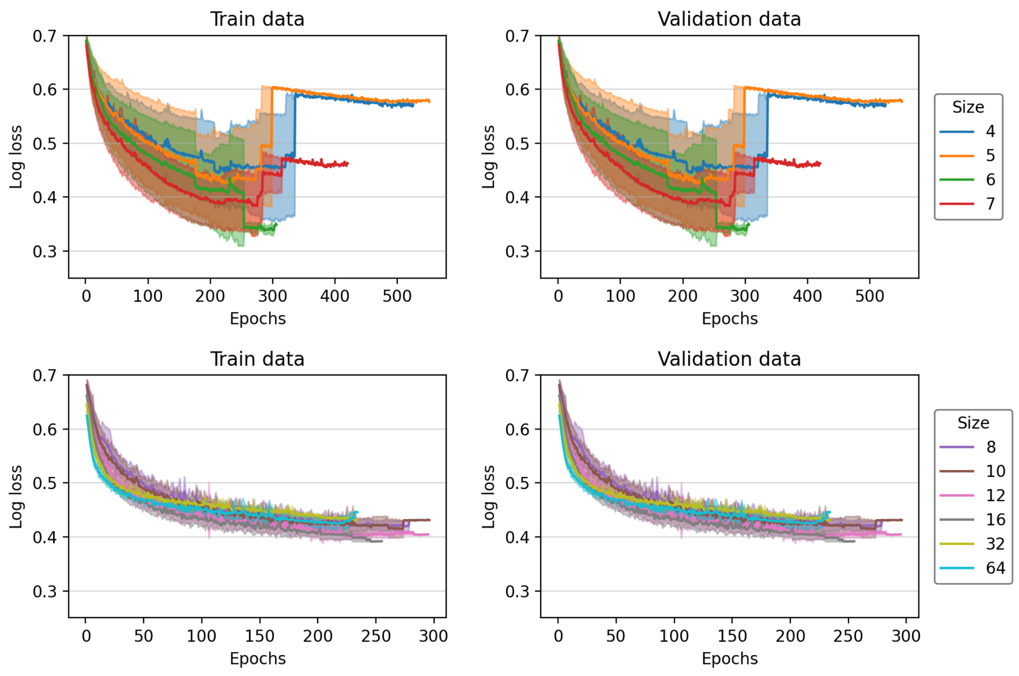


Figure 4.2: Multi-task architecture exhibits very unstable training when the latent size is too low. The upper two figures illustrate the log loss on unstable multi-task models, while the lower two figures present the log loss on stable multi-task models. The color in the legend represents the size of the latent layer of the backbone of the model. It can be observed that, due to less stable training than autoencoder models, there is no clear indication as to which multi-task model performs the best, as their metrics have a considerable overlap.

Upon closer examination, it became evident that for each unstable multi-task model there were at least two instances of failed training, typically those that were trained for the longest periods of time and demonstrated only slight improvements over epochs. The greatest variance in log loss and thus the most unstable training was exhibited by the 5-multitask model, while the 64-multitask model exhibited the lowest variance. There is also no clear indication from the plots of which multi-task model performs the best, as their metrics overlap considerably. On average, the 16-multitask model achieved the lowest validation log loss ($logloss = 0.406 \pm 0.008$), while the 64-multitask model achieved the highest validation AUC score ($AUC = 0.816 \pm 0.010$).

4.2 Testing

The objective of this thesis was to examine the potential of gene expression encodings for transfer learning. For this purpose, we took the trained models and used only the encoder component to generate encodings of samples from test data sets. We compared the results of models constructed on those encodings to models built on raw data, which we will refer to as the baseline. A comprehensive account of the testing results and further analysis of the encodings is available in Appendix B.

Figure 4.3 depicts the AUC-AUC plots for autoencoders and multi-task models. For each model, we plotted a trend line to assess the proximity of the test metrics to the baseline. The results demonstrated that larger models produced significantly better results than smaller models, which held for both architectures. This is supported by the trend lines being much closer to the diagonal and the results having much less variance. For both architectures, we observed that our models improved the results for data sets with a lower-performing baseline. In contrast, for data sets with a higher-performing baseline, our models performed poorer. This was in line with the findings of our previous experiments.

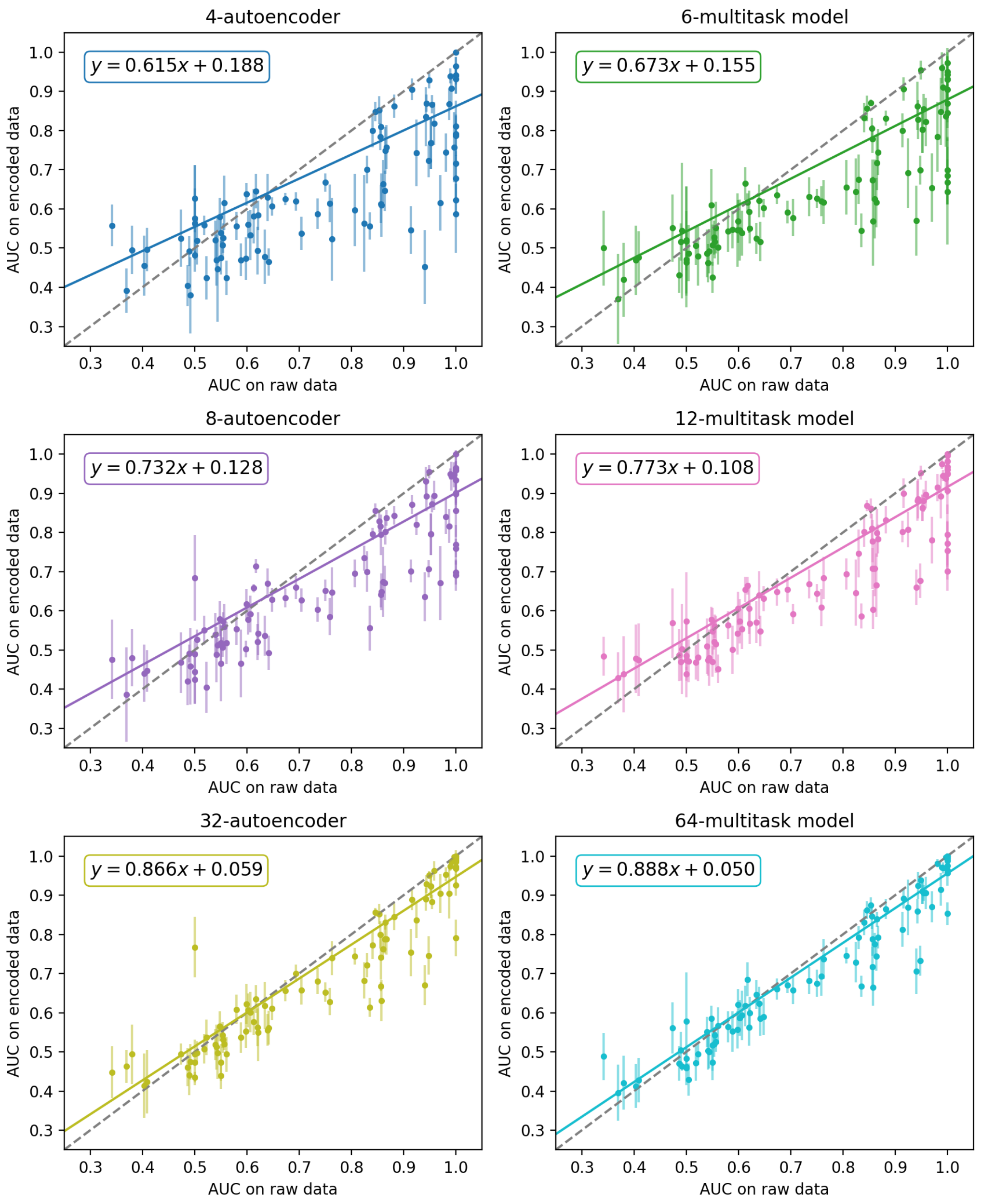


Figure 4.3: A larger encoding size produces results closer to baseline, regardless of model architecture. The figures illustrate the AUC-AUC plots of six different models from both autoencoder and multi-task architectures. As the size of the model increases, the results approach those of the baseline, as indicated by the trend line, for which the equation is provided in the upper left corner of each figure.

Figure 4.4 presents the TPR-TNR plots for autoencoders and multi-task models. A similar trend is evident in both architectures, where an increase in the size of the latent layer corresponds to an improvement in TNR. The highest TNR is achieved by the 64-autoencoder ($TNR = 0.831 \pm 0.003$) and the 64-multitask model ($TNR = 0.831 \pm 0.003$). However, the TPR appears to improve only up to a certain latent size, as evidenced by a decline in the TPR metric when the latent layer is large enough. The highest TPR in the case of autoencoders is achieved by the 16-autoencoder ($TPR = 0.653 \pm 0.007$), while in multi-task models, the highest TPR is achieved by the 32-multitask model ($TPR = 0.655 \pm 0.004$). In both cases, the highest TPR indicates that approximately one-third of the positive samples were misclassified. This indicates that a larger latent size initially helps with identifying both positive and negative samples, but a too-large latent size results in a considerable number of false negatives in the search for additional negative samples. It is noteworthy that the TPR metric varies only from 0.628 to 0.655, while TNR varies from 0.671 to 0.831. This indicates that the improvement in TPR for larger models is considerably less pronounced than the improvement in TNR. This discrepancy may be attributed to the imbalanced datasets, as the positive samples represent only a quarter of the testing samples, and identifying the majority class appears to be the most effective strategy for logistic regression models.

In the final stage of the analysis, we analysed the encodings. To this end, we compared the PCA projections of the raw values and the encoded values of the test data sets. Figure 4.5 presents the PCA projection plots of the raw values and the encodings of different models. A clear division between the data sets is evident in the PCA projection of the raw values (Figure 4.5a), indicating that their gene expression values are inherently different. The two data sets that are particularly prominent on the far right side are GDS4267 (in green) and GDS5205 (in yellow), which contain data on peripheral blood gene expressions in patients with juvenile idiopathic arthritis and survival of glioblastoma, respectively. There are other peripheral blood gene expression

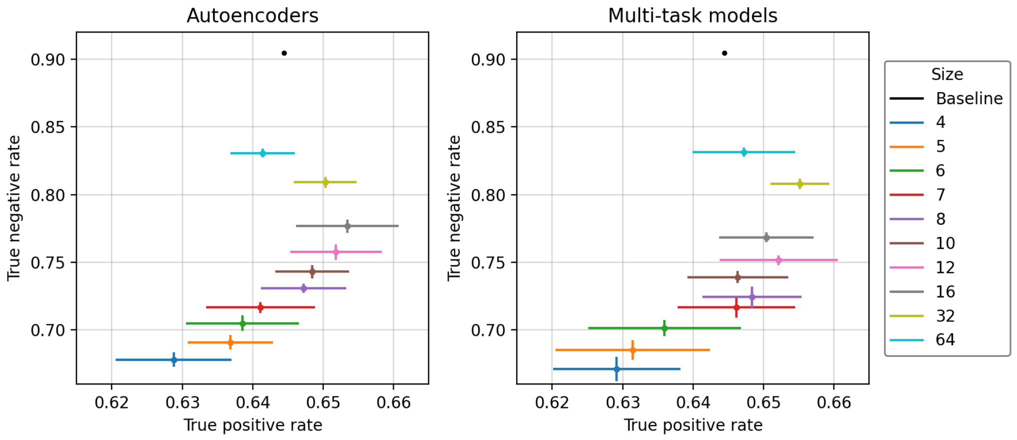


Figure 4.4: Encodings help identify positive, but not negative samples. The figures illustrate TPR-TNR plots for autoencoder (left) and multi-task architectures (right). Baseline (black) has the highest TNR rate in both cases, but many models in both architectures have better TPR scores.

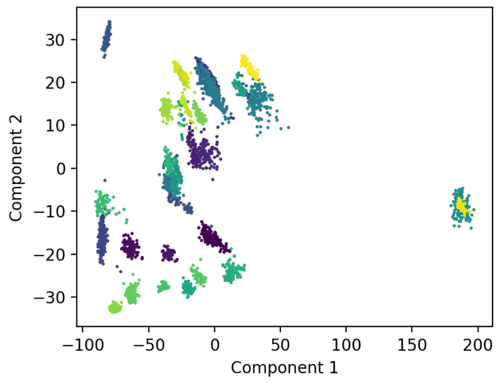
and glioblastoma data sets, and the reason for their separation from the rest is unknown to us.

Upon examination of the PCA projections of the encoded data from the autoencoder models, it is evident that there is no significant difference from the PCA projection of the raw values. The PCA projections of both the 7-autoencoder and 64-autoencoder encodings (Figure 4.5b, upper two plots) exhibit a comparable data structure to that of the raw data PCA projection and produce the same clear division between GDS4267, GDS5205 and the rest of the data sets. This phenomenon is observed in all autoencoder models, irrespective of their latent size The only differences in the PCA plots appear to be rotation, slight translation, and/or reflection of the projected samples. The 7-autoencoder model displays reflection, while the 64-autoencoder exhibits rotation. This indicates that the autoencoder models preserve the shape and relation between samples of the original data and manage to reduce their dimensionality without losing spatial information.

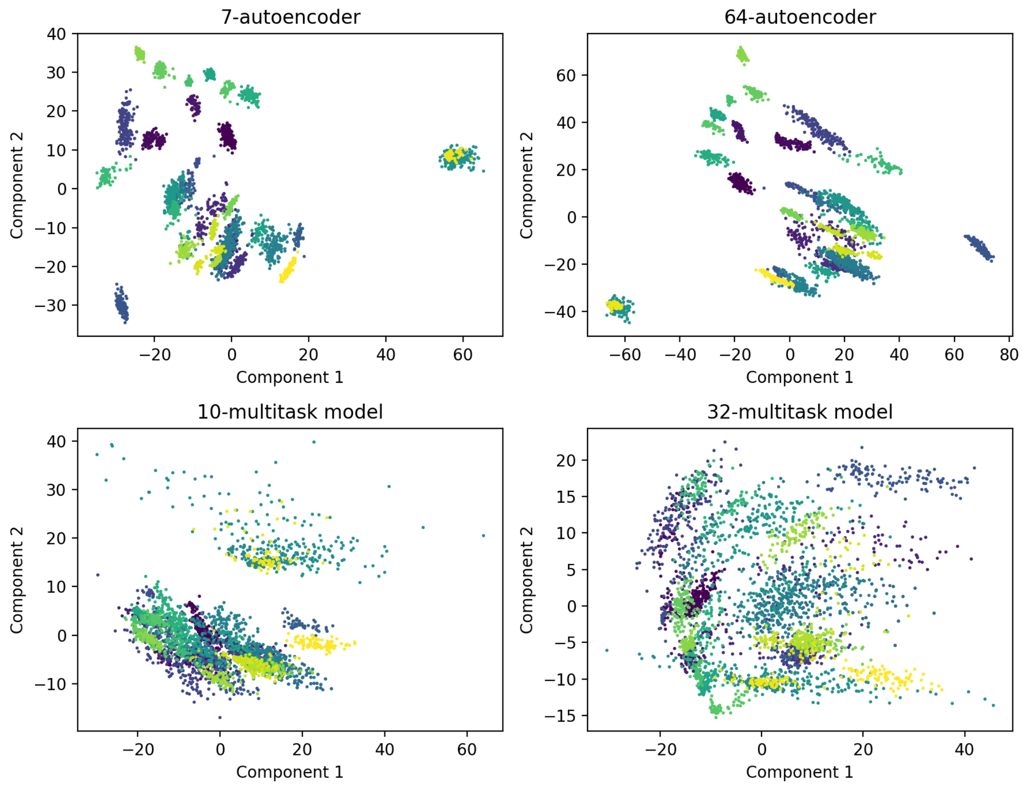
In contrast to the autoencoder models, the situation is different with multi-task models, where there are two different PCA projection cases. In the first case, the data sets GDS4267 and GDS5205 are still distinguished from the remaining data sets, while the latter are projected to a smaller cluster. This is the case with the 10-multitask model (Figure 4.5b, lower left), the majority of the smaller models (4-, 5-, 6-, 7-, and 8-multitask models), and unexpectedly, with the 64-multitask model, although there the separation of data sets is less pronounced. The second case is characterised by the projection of all data to a smaller region in a more homogeneous manner. This is the case with the 32-multitask model (Figure 4.5b, lower right) and other larger models (10-, 12-, and 16-multitask models). Interestingly, the observed projection pattern seems to correlate with the stability of the training process. In this regard, smaller models that exhibited highly unstable training metrics across different runs produce PCA projections that are characterised by the first case, and vice versa.

Upon examination of the explained variance of the PCA components, several noteworthy observations emerge. The PCA projection of the raw values indicates that a single component is sufficient to explain 80.8% of the variance. Four components account for more than 90% of the variance, while 13 components account for more than 95% of the variance. In total, only five components account for more than 1% individually, 22 components account for more than 0.1%, and 98 components account for more than 0.01%. It is noteworthy that both the data and the PCA projections have 884 components, indicating that the data resides on a hyperplane and is highly compressible.

In the context of autoencoder models, we observed that the larger the model, the greater the number of components required to reach a certain threshold of explained variance. Nevertheless, the average number of components required to achieve at least 50% explained variance is two. For a 7-autoencoder, this equates to 1.7 ± 0.5, while for a 64-autoencoder, it is 2.0 ± 0.0, with the 4-autoencoder representing the exception with 1.3 ± 0.5. The first component accounts for $52.2\% \pm 4.6\%$ of the variance for the 4-autoencoder, and only $35.9\% \pm 1.7\%$ for the 64-autoencoder, with the



(a) PCA projection of raw samples from test data sets.



(b) PCA projection of encoded samples from test data sets.

Figure 4.5: Autoencoders maintain the original data structure, whereas multi-task models do not. The PCA projections of the raw test data sets and encodings created by different models. Different colors represent different data sets. It can be observed that the samples encoded by autoencoders maintain the original data structure. In contrast, the distinction between different data sets is no longer evident in samples, encoded by multi-task models.

other models exhibiting values in between. The encodings, produced by the autoencoder models, are superior to PCA projection on raw data when preserving 95% of data variance as they require up to 7.2 ± 0.4 components instead of 13. Our observations indicate that smaller autoencoder models require fewer components to achieve a given level of explained variance than larger ones, but this can be attributed to the fact that they have only a few latent dimensions.

Again, the situation is different for multi-task models. While they still require more than one component to achieve at least 50% of explained variance on average, this number is notably smaller from the autoencoder models, ranging from 1.1 ± 0.3 for the 4-multitask model to 1.9 ± 0.3 for the 12-multitask model. This is also the case for thresholds of 90% and 95% explained variance. Furthermore, the first component exhibits a greater degree of explained variance, with the 4-multitask model explaining 63.9% ± 14.5% of variance, the 16-multitask model explaining 44.0% ± 9.1% of variance, with the other models in between. Once more, the encodings of the multitask models are superior to raw data in preserving explained variance as seven out of ten models require less than four components for achieving 90% of explained variance and require up to 6.1 ± 0.5 components instead of 13 for achieving 95% of explained variance. It is noteworthy that the number of components required to achieve a certain threshold of variance explainability initially increases but then decreases after the 16-multitask model.

Chapter 5

Conclusion

In this thesis, we have focused on gene expression data and phenotype trait prediction on small data sets. We approached the small data set problem by combining many small data sets into a larger one, thus creating a more stable learning environment for training a model that produces meaningful gene encodings for phenotype prediction. We used transfer learning to evaluate the performance of the encodings on test data sets and inspected the encodings.

We evaluated two different architectures, autoencoders and multi-task models. The autoencoder architecture proved stable during training and achieved low MSE results. The multi-task architecture was difficult to train, especially for models with smaller latent sizes. To evaluate the test data sets, we established the baseline as the performance of logistic regression on unencoded data. The performance of both architectures increased with greater encoding size. The multi-task model with a latent size 64 demonstrated the closest performance to the baseline, although it did not surpass it. We inspected the confusion matrices of both architectures and discovered that the models with larger encoding sizes achieved higher true negative rates. However, the true positive rate metric indicated that larger encoding sizes did not always improve the results. The models with encodings of either size 16 or 32 achieved the highest true positive rates, and in both architectures, the largest models performed significantly worse than the best.

When we examined the embedding space and analysed the PCA projections, we discovered that the unencoded data exhibited high compressibility, with the first component accounting for over 80% of the variance. The autoencoder architecture preserved the shape of the unencoded data, and its projections only differed from the projections of the unencoded data in either rotation or mirroring effect. In contrast, the multi-task architecture produced two different types of projections. The smaller models still somewhat retained the recognisable data structure, but exaggerated the projection of the two outlier data sets, while the larger models mixed all the data sets together.

There are a number of ways in which our work could be improved. The process that is most in need of improvement is the data selection process. The data we are dealing with has a biological nature and is highly specific, as each data set describes a different biological process from a different published study. We lack formal training in biology, and the data selection process was conducted exclusively from a computer science perspective. We selected only the data sets where we reasoned there was potential for predictable attributes, irrespective of their connection to the accompanying gene expression data. To make a more informed selection, it would be beneficial to consult with an expert, and possibly more than one, given the different specific problems each data set describes. Furthermore, we could choose a different database for the data selection process. While the GEO database is not the largest gene expression database, it is curated and has already been parsed into an easily readable format.

The analysis of the embedding space could be improved for the multi-task architecture. It is evident that the projections do not maintain the original data structure, but there may be some clusters of samples across data sets with similar characteristics that we are unable to recognise. This would suggest that the multi-task models are more suitable for creating gene expression encodings as they are able to recognise some characteristics across data sets and make them truly universal.

Bibliography

[1] P. Larranaga, B. Calvo, R. Santana, *et al.*, "Machine learning in bioinformatics," *Briefings in Bioinformatics*, vol. 7, no. 1, pp. 86–112, 2006.

[2] L. Peltonen, A. Jalanko, and T. Varilo, "Molecular Genetics the Finnish Disease Heritage," *Human Molecular Genetics*, vol. 8, no. 10, pp. 1913–1923, 1999.

[3] J. Snider, M. Kotlyar, P. Saraon, *et al.*, "Fundamentals of protein interaction network mapping," *Molecular Systems Biology*, vol. 11, no. 12, p. 848, 2015.

[4] D. M. Vo, N.-Q. Nguyen, and S.-W. Lee, "Classification of breast cancer histology images using incremental boosting convolution networks," *Information Sciences*, vol. 482, pp. 123–138, 2019.

[5] J.-J. M. Riethoven, "Regulatory Regions in DNA: Promoters, Enhancers, Silencers, and Insulators," *Computational Biology of Transcription Factor Binding*, pp. 33–42, 2010.

[6] K. P. Singh, C. Miaskowski, A. A. Dhruva, *et al.*, "Mechanisms and Measurement of Changes in Gene Expression," *Biological Research for Nursing*, vol. 20, no. 4, pp. 369–382, 2018.

[7] P. Wu, Q. Feng, V. E. Kerchberger, *et al.*, "Integrating gene expression and clinical data to identify drug repurposing candidates for hyperlipidemia and hypertension," *Nature Communications*, vol. 13, no. 1, p. 46, 2022.

[8] A. Athar, A. Füllgrabe, N. George, *et al.*, "ArrayExpress update–from bulk to single-cell expression data," *Nucleic Acids Research*, vol. 47, no. D1, pp. D711–D715, 2019.

[9] M. A. Merideth, L. B. Gordon, S. Clauss, *et al.*, "Phenotype and Course of Hutchinson–Gilford Progeria Syndrome," *New England Journal of Medicine*, vol. 358, no. 6, pp. 592–604, 2008.

[10] D. G. Beer, S. L. Kardia, C.-C. Huang, *et al.*, "Gene-expression profiles predict survival of patients with lung adenocarcinoma," *Nature Medicine*, vol. 8, no. 8, pp. 816–824, 2002.

[11] F. Baty, M. Facompré, J. Wiegand, *et al.*, "Analysis with respect to instrumental variables for the exploration of microarray data structures," *BMC Bioinformatics*, vol. 7, pp. 1–8, 2006.

[12] A. Schramm, B. Schowe, K. Fielitz, *et al.*, "Exon-level expression analyses identify MYCN and NTRK1 as major determinants of alternative exon usage and robustly predict primary neuroblastoma outcome," *British Journal of Cancer*, vol. 107, no. 8, pp. 1409–1417, 2012.

[13] C. Park, J. Ha, and S. Park, "Prediction of Alzheimer's disease based on deep neural network by integrating gene expression and DNA methylation dataset," *Expert Systems with Applications*, vol. 140, p. 112873, 2020.

[14] S. Bozinovski and A. Fulgosi, "The influence of pattern similarity and transfer learning upon training of a base perceptron B2," in *Proceedings of Symposium Informatica*, vol. 3, pp. 121–126, 1976.

[15] M. Hussain, J. J. Bird, and D. R. Faria, "A Study on CNN Transfer Learning for Image Classification," in *Advances in Computational Intelligence Systems*, vol. 840, pp. 191–202, Springer, 2018.

[16] J. Howard and S. Ruder, "Universal Language Model Fine-tuning for Text Classification," *arXiv Preprint*, 2018.

[17] C. T. Choy, C. H. Wong, and S. L. Chan, "Embedding of Genes Using Cancer Gene Expression Data: Biological Relevance and Potential Application on Biomarker Discovery," *Frontiers in Genetics*, vol. 9, p. 682, 2019.

[18] J. Du, P. Jia, Y. Dai, *et al.*, "Gene2vec: distributed representation of genes based on co-expression," *BMC Genomics*, vol. 20, pp. 7–15, 2019.

[19] C.-Y. Cheng, Y. Li, K. Varala, *et al.*, "Evolutionarily informed machine learning enhances the power of predictive gene-to-phenotype relationships," *Nature Communications*, vol. 12, no. 1, p. 5627, 2021.

[20] J. Chen, X. Wang, A. Ma, *et al.*, "Deep transfer learning of cancer drug responses by integrating bulk and single-cell RNA-seq data," *Nature Communications*, vol. 13, no. 1, p. 6494, 2022.

[21] R. Edgar, M. Domrachev, and A. E. Lash, "Gene Expression Omnibus: NCBI gene expression and hybridization array data repository," *Nucleic Acids Research*, vol. 30, no. 1, pp. 207–210, 2002.

[22] A. Subramanian, R. Narayan, S. M. Corsello, *et al.*, "A next generation Connectivity Map: L1000 platform and the first 1,000,000 profiles," *Cell*, vol. 171, no. 6, pp. 1437–1452.e17, 2017.

[23] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv Preprint*, 2014.

[24] B. Alberts, A. Johnson, J. Lewis, *et al.*, "Analyzing Protein Structure and Function," in *Molecular Biology of the Cell. 4th edition*, pp. 1331–1356, Garland Science, 2002.

[25] D. J. Hunter, "Gene–environment interactions in human diseases," *Nature Reviews Genetics*, vol. 6, no. 4, pp. 287–298, 2005.

[26] C. Buccitelli and M. Selbach, "mRNAs, proteins and the emerging principles of gene expression control," *Nature Reviews Genetics*, vol. 21, no. 10, pp. 630–644, 2020.

[27] R. Govindarajan, J. Duraiyan, K. Kaliyappan, and M. Palanisamy, "Microarray and its applications," *Journal of Pharmacy and Bioallied Sciences*, vol. 4, no. Suppl 2, pp. S310–S312, 2012.

[28] R. Stark, M. Grzelak, and J. Hadfield, "RNA sequencing: the teenage years," *Nature Reviews Genetics*, vol. 20, no. 11, pp. 631–656, 2019.

[29] The UniProt Consortium, "UniProt: a worldwide hub of protein knowledge," *Nucleic Acids Research*, vol. 47, no. D1, pp. D506–D515, 2018.

[30] G.-Y. Chao, T.-I. Tsai, T.-J. Lu, *et al.*, "A new approach to prediction of radiotherapy of bladder cancer cells in small dataset analysis," *Expert Systems with Applications*, vol. 38, no. 7, pp. 7963–7969, 2011.

[31] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, p. 321–357, 2002.

[32] A. Hawkins-Hooker, F. Depardieu, S. Baur, *et al.*, "Generating functional protein variants with variational autoencoders," *PLoS Computational Biology*, vol. 17, no. 2, p. e1008736, 2021.

[33] D. Cottrell, P. S. Swain, and P. F. Tupper, "Stochastic branching-diffusion models for gene expression," *Proceedings of the National Academy of Sciences*, vol. 109, no. 25, pp. 9699–9704, 2012.

[34] U. Sarkans, M. Gostev, A. Athar, *et al.*, "The BioStudies database—one stop shop for all data supporting a life sciences study," *Nucleic Acids Research*, vol. 46, no. D1, pp. D1266–D1270, 2018.

[35] P. Danecek, A. Auton, G. Abecasis, *et al.*, "The variant call format and VCFtools," *Bioinformatics*, vol. 27, no. 15, pp. 2156–2158, 2011.

[36] S. Purcell, B. Neale, K. Todd-Brown, *et al.*, "PLINK: a tool set for whole-genome association and population-based linkage analyses," *The American Journal of Human Genetics*, vol. 81, no. 3, pp. 559–575, 2007.

[37] R. Tibshirani, T. Hastie, B. Narasimhan, and G. Chu, "Diagnosis of multiple cancer types by shrunken centroids of gene expression," *Proceedings of the National Academy of Sciences*, vol. 99, no. 10, pp. 6567–6572, 2002.

[38] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[39] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv Preprint*, 2013.

[40] B. Hanczar, V. Bourgeais, and F. Zehraoui, "Assessment of deep learning and transfer learning for cancer prediction based on gene expression data," *BMC Bioinformatics*, vol. 23, no. 1, p. 262, 2022.

[41] A. Torrente, M. Lukk, V. Xue, *et al.*, "Identification of cancer related genes using a comprehensive map of human gene expression," *PloS One*, vol. 11, no. 6, p. e0157484, 2016.

[42] J. Demšar, T. Curk, A. Erjavec, *et al.*, "Orange: Data Mining Toolbox in Python," *Journal of Machine Learning Research*, vol. 14, no. 71, pp. 2349–2353, 2013.

[43] K. Pearson, "LIII. On Lines and Planes of Closest Fit to Systems of Points in Space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[45] R. S. Zemel and G. Hinton, “Autoencoders, minimum description length and Helmholtz free energy,” in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, pp. 3–10, Citeseer, 1993.

[46] C. Eckart and G. Young, “A principal axis transformation for non-Hermitian matrices,” *Bulletin of the American Mathematical Society*, vol. 45, no. 2, pp. 118–121, 1939.

[47] S. Balakrishnama and A. Ganapathiraju, “Linear discriminant analysis - a brief tutorial,” *Institute for Signal and Information Processing*, vol. 18, no. 1998, pp. 1–8, 1998.

[48] M. A. Kramer, “Nonlinear principal component analysis using autoassociative neural networks,” *AIChE Journal*, vol. 37, no. 2, pp. 233–243, 1991.

[49] L. Yasenko, Y. Klyatchenko, and O. Tarasenko-Klyatchenko, “Image noise reduction by denoising autoencoder,” in *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pp. 351–355, IEEE, 2020.

[50] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, “Autoencoder-based network anomaly detection,” in *2018 Wireless Telecommunications Symposium (WTS)*, pp. 1–5, IEEE, 2018.

[51] A. P. S. Chandar, S. Lauly, H. Larochelle, *et al.*, “An autoencoder approach to learning bilingual word representations,” in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, vol. 27, pp. 1853—1861, MIT Press, 2014.

[52] H. Bilen and A. Vedaldi, “Universal representations: The missing link between faces, text, planktons, and cat breeds,” *arXiv Preprint*, 2017.

[53] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Learning multiple visual domains with residual adapters," in *Advances in Neural Information Processing Systems*, vol. 30, pp. 506–516, 2017.

[54] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32, pp. 8026–8037, 2019.

[55] N. S. Detlefsen, J. Borovec, J. Schock, *et al.*, "TorchMetrics - Measuring Reproducibility in PyTorch," *Journal of Open Source Software*, vol. 7, no. 70, p. 4101, 2022.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[57] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[58] J. Reback, W. McKinney, J. Van Den Bossche, *et al.*, "pandas-dev/pandas: Pandas," *Zenodo*, 2020.

Appendix A

Detailed train results

This chapter presents a more detailed examination of the training results, namely final metric values and train times. To compare the models' performance, we selected two at a time by generating a million random samples with average results as the mean and corresponding standard deviation. We compared them pairwise, and empirically determined the percentage of one model achieving a better score than the other.

The final average autoencoder training and validation metrics with standard deviation are presented in Table A.1. We can observe a trend that an increase in the latent size is associated with lower MSE loss and higher R^2 coefficient. When accounting for uncertainty, the 64-autoencoder outperforms the 32-autoencoder with a probability of 61.5% for train MSE, 72.8% for validation MSE, 54.3% for train R^2 and 67.2% for validation R^2. The validation MSE for the 64-autoencoder was 135.1 ± 3.3, and with 884 gene expression inputs, this means an average squared error of 0.1528 ± 0.0037 or an average error of 0.3909 ± 0.0048 per gene expression. Given that the gene expressions in our data sets have values of around 10, this error is relatively minor. The same model achieved a validation R^2 value of 0.983 ± 0.001, indicating that it can explain $98.3\% \pm 0.1\%$ of the output variance.

The final average multi-task model training and validation metrics with standard deviation are presented in Table A.2. We can observe that an in-

	MSE		R^2	
Size	Train	Validation	Train	Validation
4	186.7 ± 6.7	202.1 ± 5.2	0.9757 ± 0.0012	0.9747 ± 0.0009
5	164.6 ± 5.4	183.9 ± 3.9	0.9785 ± 0.0008	0.9769 ± 0.0007
6	159.2 ± 4.3	178.1 ± 2.1	0.9790 ± 0.0016	0.9777 ± 0.0003
7	151.1 ± 5.5	171.2 ± 3.4	0.9802 ± 0.0006	0.9784 ± 0.0006
8	145.4 ± 4.7	165.6 ± 3.3	0.9809 ± 0.0008	0.9792 ± 0.0005
10	136.6 ± 4.7	158.5 ± 3.5	0.9820 ± 0.0010	0.9800 ± 0.0003
12	130.4 ± 4.4	151.9 ± 3.0	0.9824 ± 0.0010	0.9808 ± 0.0006
16	125.2 ± 3.5	138.5 ± 2.6	0.9832 ± 0.0007	0.9826 ± 0.0005
32	112.8 ± 0.9	137.2 ± 1.0	0.9849 ± 0.0007	0.9828 ± 0.0003
64	*111.6 ± 4.0*	*135.1 ± 3.3*	*0.9850 ± 0.0006*	*0.9831 ± 0.0006*

Table A.1: The final MSE and R^2 values on train and validation data for the autoencoder models. The values represent the mean with standard deviation. The best results are highlighted in *italic*.

crease in the latent size results in an improvement in both the log loss and the AUC of the model. When accounting for uncertainty, the 64-multitask model outperforms the 16-multitask model with a probability of 78.7% for train log loss. In contrast, the 16-multitask model outperforms the 12-multitask model with a probability of 55.5% for validation log loss. Similarly, the 64-multitask model outperforms the 32-multitask model with a probability of 75.9% for train AUC and 80.3% for validation AUC. The average validation log loss of the 16-multitask model was 0.406 $\pm$ 0.008, indicating that the model attributed the probability of 66.6% $\pm$ 0.5% on average to the correct class, which is not that high. Nevertheless, even the 4-multitask model, which has the highest validation log loss score, attributed the probability of 60.4% $\pm$ 3.9% on average to the correct class, which is superior to random guessing. The 64-multitask model achieved the highest validation AUC score of 0.816 $\pm$ 0.010, indicating that it can differentiate between a positive and negative sample 81.6% $\pm$ 1.0% of the time.

Table A.3 presents the mean epoch time in seconds for all autoencoder and multi-task models. The epoch times differ minimally between architectures and latent sizes. We anticipated to observe greater differences, but as we inspected the number of parameters per model, which range from 717,168 to 753,408 for multi-task models and from 1,433,600 to 1,495,040 for autoencoders, the lack of epoch time variation within model architecture should have been expected. The lack of epoch times differences between the autoencoder and multi-task architecture surprised us, given that the latter is half the size of the former. The multi-task architecture demonstrated faster training, but slightly slower validation epoch times than the autoencoder architecture. This indicates that the bottleneck in training and validation was not the forward pass or backward propagation, but the size of the data. The slower validation time observed in multi-domain architectures can be attributed to the process of task masking for each sample. We identified two exceptions: the 32-autoencoder and the 64-autoencoder. The epoch time for 32-autoencoder varies considerably, ranging from 0.6 to 1.7 seconds across all

	log loss		AUC	
Size	Train	Validation	Train	Validation
4	0.458 ± 0.072	0.507 ± 0.064	0.720 ± 0.068	0.670 ± 0.060
5	0.442 ± 0.093	0.493 ± 0.084	0.757 ± 0.085	0.694 ± 0.080
6	0.421 ± 0.093	0.479 ± 0.082	0.768 ± 0.089	0.712 ± 0.076
7	0.380 ± 0.039	0.443 ± 0.032	0.806 ± 0.040	0.735 ± 0.039
8	0.357 ± 0.018	0.424 ± 0.010	0.832 ± 0.017	0.758 ± 0.016
10	0.349 ± 0.020	0.422 ± 0.012	0.845 ± 0.018	0.767 ± 0.017
12	0.340 ± 0.025	0.408 ± 0.012	0.855 ± 0.022	0.783 ± 0.022
16	0.339 ± 0.016	*0.406 ± 0.008*	0.857 ± 0.013	0.783 ± 0.019
32	0.340 ± 0.016	0.444 ± 0.007	0.859 ± 0.015	0.800 ± 0.016
64	*0.322 ± 0.014*	0.430 ± 0.009	*0.873 ± 0.013*	*0.816 ± 0.010*

Table A.2: The final log loss and AUC values on train and validation data for the multi-task models. The values represent the mean with standard deviation. The best results are highlighted in *italic*.

	Autoencoder		Multi-task model	
Size	Train time [s]	Val time [s]	Train time [s]	Val time [s]
4	0.286 ± 0.025	0.012 ± 0.001	0.257 ± 0.023	0.013 ± 0.002
5	0.283 ± 0.023	0.012 ± 0.001	0.250 ± 0.017	0.013 ± 0.001
6	0.283 ± 0.021	0.012 ± 0.001	0.255 ± 0.019	0.013 ± 0.001
7	0.286 ± 0.025	0.012 ± 0.001	0.248 ± 0.019	0.013 ± 0.001
8	0.282 ± 0.023	0.012 ± 0.002	0.252 ± 0.023	0.013 ± 0.002
10	0.285 ± 0.025	0.012 ± 0.001	0.247 ± 0.017	0.013 ± 0.001
12	0.283 ± 0.025	0.012 ± 0.001	0.248 ± 0.016	0.013 ± 0.001
16	0.281 ± 0.025	0.012 ± 0.002	0.249 ± 0.020	0.013 ± 0.002
32	1.310 ± 0.408	0.016 ± 0.004	0.254 ± 0.021	0.013 ± 0.002
64	0.442 ± 0.152	0.015 ± 0.004	0.253 ± 0.051	0.013 ± 0.001

Table A.3: The mean train epoch times for the autoencoder and multi-task models. Both architectures have relatively low training and validation (denoted Val) times except for 32-autoencoder and 64-autoencoder models, the deviation of which we cannot explain.

10 runs. Similar is true for the 64-autoencoder, with an epoch time of approximately 0.4 seconds. These exceptions indicate that the training process may have been affected by the concurrent execution of another resource-intensive program, which slowed the training process considerably.

Table A.4 presents the average number of training epochs for all autoencoder and multi-task models. It can be observed that multi-task models have trained for considerably fewer epochs than autoencoders. Furthermore, it appears that the larger the multi-task model, the fewer training epochs it underwent. The epoch numbers for autoencoders align with our expectations, as larger models were trained for longer. Low epoch numbers for multi-task models may indicate problems with training, such as the optimizer's learning rate and early stopping criteria. It is also possible that the small number of samples for each prediction head contributed to this outcome.

	Autoencoder	Multi-task model
Size	Epochs	Epochs
4	413.0 ± 40.4	278.9 ± 94.5
5	447.7 ± 62.0	265.8 ± 104.4
6	413.9 ± 31.4	243.4 ± 42.0
7	422.9 ± 44.9	273.0 ± 58.7
8	445.2 ± 35.7	239.4 ± 29.6
10	466.5 ± 55.8	241.7 ± 28.8
12	514.9 ± 53.3	241.1 ± 32.8
16	552.9 ± 45.9	225.2 ± 17.7
32	605.1 ± 18.9	201.2 ± 20.5
64	593.6 ± 49.4	211.0 ± 17.7

Table A.4: The mean number of training epochs for autoencoder and multi-task models. Autoencoders require more epochs the larger the model, while this is not the case for multi-task models.

Appendix B

Detailed test results

This chapter presents a more detailed examination of the testing results, namely performance on the testing data sets and the analysis of the encodings. To compare the models, we selected two at a time by generating a million random samples with average results as the mean and corresponding standard deviation. We compared them pairwise, and empirically determined the percentage of one model achieving a better score than the other.

The slope coefficients and interceptions of the testing trend lines are presented in Table B.1. As the size of the model encoding increases, the model's performance approaches that of the baseline. It is evident that for smaller sizes, the autoencoder architecture outperforms the multi-task models, as evidenced by the higher slope and lower intercept. However, the 64-multitask model achieved the highest slope and lowest intercept value, demonstrating a higher slope than the 64-autoencoder with a probability of 63.8% and a lower intercept value then the 64-autoencoder with a probability of 60.9%. With regard to the autoencoder architecture, the 32-autoencoder achieved the same intercept value as the 64-autoencoder, but due to higher variance and lower slope coefficient, we consider it inferior.

The Table B.2 presents the results of the TPR and TNR metrics. The TNR metric steadily increases with the size of the encodings, indicating that larger encoding sizes are beneficial for the identification of negative samples.

	Autoencoder		Multi-task model	
Size	slope	intercept	slope	intercept
4	0.615 ± 0.051	0.188 ± 0.039	0.564 ± 0.043	0.223 ± 0.034
5	0.650 ± 0.046	0.168 ± 0.035	0.611 ± 0.045	0.195 ± 0.035
6	0.691 ± 0.044	0.145 ± 0.034	0.673 ± 0.042	0.155 ± 0.032
7	0.753 ± 0.042	0.104 ± 0.033	0.692 ± 0.041	0.153 ± 0.032
8	0.732 ± 0.041	0.128 ± 0.032	0.724 ± 0.038	0.133 ± 0.030
10	0.771 ± 0.041	0.105 ± 0.032	0.755 ± 0.037	0.115 ± 0.029
12	0.785 ± 0.039	0.101 ± 0.030	0.773 ± 0.037	0.108 ± 0.028
16	0.810 ± 0.036	0.089 ± 0.028	0.804 ± 0.037	0.090 ± 0.029
32	0.866 ± 0.035	0.059 ± 0.027	0.842 ± 0.031	0.080 ± 0.024
64	*0.873 ± 0.033*	*0.059 ± 0.025*	*0.888 ± 0.027*	*0.050 ± 0.021*

Table B.1: The trend line parameters for AUC-AUC plots for autoencoder and multi-task architectures. Larger latent size leads to closer results to the baseline, however neither architecture is able to surpass it. The best values for each architecture are highlighted in *italic*.

	Autoencoder		Multi-task model	
Size	TPR	TNR	TPR	TNR
4	0.629 ± 0.008	0.678 ± 0.005	0.629 ± 0.009	0.671 ± 0.009
5	0.637 ± 0.006	0.691 ± 0.005	0.631 ± 0.011	0.685 ± 0.007
6	0.639 ± 0.008	0.705 ± 0.006	0.636 ± 0.011	0.702 ± 0.006
7	0.641 ± 0.008	0.717 ± 0.004	0.646 ± 0.008	0.717 ± 0.008
8	0.647 ± 0.006	0.731 ± 0.003	0.648 ± 0.007	0.724 ± 0.008
10	0.648 ± 0.005	0.743 ± 0.005	0.646 ± 0.007	0.739 ± 0.004
12	0.652 ± 0.007	0.758 ± 0.006	0.652 ± 0.008	0.752 ± 0.004
16	*0.653 ± 0.007*	0.777 ± 0.005	0.650 ± 0.007	0.768 ± 0.004
32	0.650 ± 0.004	0.809 ± 0.004	*0.655 ± 0.004*	0.808 ± 0.004
64	0.641 ± 0.005	*0.831 ± 0.003*	0.647 ± 0.007	*0.831 ± 0.003*

Table B.2: TPR and TNR values for autoencoder and multi-task architectures. TNR metric steadily increases, while the TPR metric first increases, but starts to decrease after 16-autoencoder and 32-multitask model. The best values for each architecture are highlighted in *italic*.

We can observe that the changes in the TPR metric are minor and that an increase in the size of the encodings does not result in an increase in its value. This suggests that, despite the models with an encoding size of 64 achieving the best training results and being the closest to the baseline results, they are unable to consistently predict positive samples any more effectively than smaller models. The 64-autoencoder and 64-multitask model achieved the highest TNR metric, while the highest TPR was achieved by the 32-multitask model, which outperformed the 16-autoencoder model with a probability of 59.8%.

The number of components required to reach certain explained variance thresholds is presented in Table B.3. We can observe that in the case of the autoencoder architecture, we require a greater number of components to achieve the same level of explained variance as the size of the encodings

	Autoencoder			Multi-task model		
Size	50%	90%	95%	50%	90%	95%
4	1.3 ± 0.5	3.0 ± 0.0	3.6 ± 0.5	1.1 ± 0.3	2.4 ± 0.7	3.0 ± 0.6
5	1.8 ± 0.4	3.1 ± 0.3	4.1 ± 0.3	1.2 ± 0.4	2.5 ± 0.8	3.3 ± 1.0
6	1.9 ± 0.3	3.3 ± 0.5	4.0 ± 0.0	1.6 ± 0.5	2.8 ± 1.1	3.2 ± 1.2
7	1.7 ± 0.5	3.7 ± 0.5	4.8 ± 0.4	1.5 ± 0.5	3.4 ± 0.7	4.2 ± 0.9
8	1.8 ± 0.4	3.9 ± 0.3	5.0 ± 0.0	1.8 ± 0.4	3.7 ± 0.5	4.7 ± 0.5
10	1.9 ± 0.3	4.2 ± 0.4	5.3 ± 0.5	1.3 ± 0.5	3.7 ± 0.5	4.9 ± 0.5
12	2.0 ± 0.0	4.3 ± 0.5	5.8 ± 0.4	1.9 ± 0.3	4.6 ± 0.7	5.9 ± 0.8
16	2.0 ± 0.0	4.5 ± 0.5	6.1 ± 0.3	1.7 ± 0.5	4.9 ± 0.5	6.1 ± 0.5
32	2.0 ± 0.0	5.0 ± 0.0	6.9 ± 0.3	1.4 ± 0.5	4.2 ± 0.6	5.7 ± 0.5
64	2.0 ± 0.0	5.3 ± 0.5	7.2 ± 0.4	1.5 ± 0.5	3.7 ± 0.5	4.8 ± 0.4
raw	1	4	13	1	4	13

Table B.3: The number of PCA components needed to achieve 50%, 90%, and 95% of explained variance. The last row represents the number of the PCA components on raw values.

increases. This is not the case for a multi-task architecture, as initially, the number of components increases, but then decreases. The 12-multitask model requires the greatest number of components to achieve 50% of explained variability, whereas the 16-multitask model requires the greatest number of components to achieve both 90% and 95% of explained variability. Both architectures outperform the PCA on the raw data in terms of the number of components required for 95% of explained variability, although this may be attributed to the limited number of available components.

Table B.4 presents the average explained variance of the first two components of the PCA projections of both autoencoders and multi-task models. It can be observed that for both architectures, the magnitude of the first components decreases as the size of the encodings increases. The multi-task models demonstrate a greater level of explained variance than the autoen-

	Autoencoder		Multi-task model	
Size	Component 1	Component 2	Component 1	Component 2
4	0.522 ± 0.046	0.284 ± 0.050	0.639 ± 0.146	0.258 ± 0.105
5	0.488 ± 0.050	0.310 ± 0.043	0.648 ± 0.178	0.208 ± 0.106
6	0.480 ± 0.020	0.307 ± 0.031	0.595 ± 0.208	0.238 ± 0.132
7	0.486 ± 0.029	0.273 ± 0.018	0.549 ± 0.106	0.235 ± 0.077
8	0.475 ± 0.031	0.282 ± 0.022	0.459 ± 0.041	0.251 ± 0.040
10	0.462 ± 0.032	0.272 ± 0.035	0.540 ± 0.094	0.240 ± 0.076
12	0.432 ± 0.022	0.286 ± 0.019	0.443 ± 0.070	0.257 ± 0.053
16	0.433 ± 0.017	0.282 ± 0.030	0.440 ± 0.091	0.221 ± 0.057
32	0.388 ± 0.009	0.298 ± 0.021	0.518 ± 0.072	0.230 ± 0.029
64	0.359 ± 0.017	0.299 ± 0.016	0.510 ± 0.047	0.256 ± 0.027
raw	0.808	0.051	0.808	0.051

Table B.4: Explained variability of the first and second component. The last row represents the explained variability of components from PCA on raw values.

coders for a given size. The second component appears to exhibit a constant explained variability and is not dependent on the encoding size. However, it is greater in autoencoders than in multi-task models. When compared to the components of the PCA on raw data, the first component, produced by either model, does not achieve anywhere near the same level of explained variability. The drop in explained variability between the first and second component of the PCA on raw data indicates that the data lies mostly on a line, while both autoencoders and multi-task models found encodings with multiple important dimensions.