DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Profiling cellular states from images using deep neural networks

Author: Michael Bornholdt

Supervisors: Prof. Dr. Daniel Rückert, Technical University of Munich

Juan Caicedo, Broad Insititute of MIT and Harvard

Shantanu Singh, Broad Insititute of MIT and Harvard

Submission Date: 12^{th} of November 2021

Statement of Originality

I, Michael Bornholdt (Student ID Number: 03722393), confirm that the version of my master's thesis in informatics I have sent to coordinators@cse.tum.de on the 12th of November 2021 is the final version of my thesis that is to be graded. With the submission of my master's thesis, I confirm that this thesis is my own work and I have documented all sources and material used.

Boston, MA, 12^{th} of November 2021 Michael Bornholdt

Acknowledgments

First of all, I would like to thank my supervisors Shantanu Singh and Juan Caicedo, both of the Broad Institute of MIT and Harvard, and Professor Dr. Daniel Rückert, of the Technical University of Munich. From the moment I met them in October of 2020, my supervisors were kind, insightful and invested in my completion of a successful Master thesis.

I want to thank Shantanu for engaging in my project so thoroughly and for sharing his deep understanding of so many processes and details from the mathematics of the evaluation metrics to the landscape of relevant papers. I want to thank Juan for always being quick to help on all questions related to DeepProfiler and deep learning in general. And finally, I would like to thank Professor Rückert for supporting my thesis abroad and for being available for advice, even from afar.

I would like to thank the whole Imaging Platform for welcoming me at the Broad Institute and being responsive and friendly at all times. Special thanks goes to Anne and Shantanu for allowing me to come in the first place and managing such an amazing team. Equal thanks go to Niranj, Greg, and Nikita for being a larger part of my project and collaborating on code updates.

Additionally, I am grateful to all the new friends I have found in Boston that made my summer most enjoyable and I hope we can stay in contact. Shout-outs go - in no particular order - to Maryam, Doreen, Britta, Fiete, Tony, Cassie, Marie(s), Alex, Frederike, and Matt - love you guys!

I want to thank my mother, Andrea Bornholdt, for proof-reading and improving various structural, grammatical components of this thesis - as has been the tradition for my academic papers.

Thanks to all and I hope I see everyone again - as we say in German: “Man sieht sich immer zweimal im Leben”

Michael Bornholdt

Abstract

Image-based profiling - the characterization and quantification of multiple observable cellular traits - is emerging as a systematic, generalized strategy that may accelerate various steps of the drug discovery process. In my master thesis, I develop a protocol for robustly and scalably quantifying (profiling) cellular states from images using deep neural networks. The JUMP-Cell Painting consortium will use this protocol to profile more than two billion cells to generate a dataset that will be made public in 2022. I apply my pipeline to the LINCS dataset as a proxy for the much larger JUMP data still being created. The LINCS database holds Cell Painting readout images of over 100 million cells perturbed with more than 1,500 different small molecules.

First, I focus on creating baseline performance results from CellProfiler, the most common profiling software. Next, the profiles from a deep learning model (pre-)trained on the ImageNet dataset are extracted and found to outperform the CellProfiler baseline. Finally, I train convolutional neural nets with a software tool called DeepProfiler and try to create models that infer even better cell profiles. For all three strategies of profile extraction (CellProfiler, pre-trained and trained), I optimize the downstream pipeline and add additional evaluation metrics.

In my experiments, I train a single-cell classification model (using the cells' perturbation as class labels) in a weakly-supervised approach as an auxiliary task. Then, I use the internal representation space of the resulting deep learning model to profile cell images. This weakly-supervised approach and its extracted profiles are evaluated based on mechanism of action (MOA) profile similarity, i.e., the assumption that compounds with similar MOAs generate similar profiles.

I found that pre-trained models outperformed the best CellProfiler data in evaluation metrics, accessibility, and speed. The profiles extracted from the pre-trained and trained profiling strategies also required less downstream processing due to their normalized and less correlated feature space. I was further able to show that trained models improve profile extraction for the type of cells that the model was trained on, proving that I am indeed learning representations. On the LINCS dataset, I matched the performance of pre-trained models with my own trained models. Finally, I present evidence that - with future improvements - the training strategy will be able to outperform pre-trained models.

Overall, I prepared a clear strategy for the JUMP-Cell Painting consortium to follow for the extraction of over two billion cells and the creation of the JUMP database - a dataset that will be publicly shared for the scientific community. I have further recommended an extensive list of subsequent research questions and paths to investigate in the future.

Contents

1. Introduction

In my master thesis, I develop a protocol for robustly and scalably quantifying (profiling) cellular states from images using deep neural networks. The JUMP-Cell Painting consortium will use this protocol to generate a large public dataset with many applications in the drug discovery process. [1]

This introductory chapter describes the drug discovery process, particularly the screening and target-based testing processes. These are relevant to my motivation and background, as the deep learning profiling protocol I develop will be most pertinent to these pharmaceutical processes. Next, I will introduce the reader to the concept of image-based profiling and Cell Painting [1], one of the most common profiling assays. Moreover, classical feature extraction software, such as CellProfiler [2], will be contrasted with the application of machine learning in the profile extraction pipeline. The final parts of this introduction will then cover downstream processing and industry applications of image-based cell profiling.

To allow for a quicker overview and better understanding, I have added subtitles in *italic* to relevant paragraphs.

1.1. The drug discovery process

Introduction to drug discovery
The drug discovery process aims to identify therapeutically valuable methods in curing and treating a disease or dysfunction. Often, the discovery process can be categorized into a preclinical and a clinical part. The former part involves a series of steps including, but not limited to, identifying, synthesizing, optimizing, characterizing, and validating candidate therapeutics, typically chemical compounds known as small molecules. A range of clinical phases then take the most promising candidates to test on animals and humans to ensure clinical efficacy and safety, among other factors.

The average cost of developing novel targets (so-called *New Molecular Entities* or *New Biological Entities)* is estimated to be $2.6 billion [3]. Only some of the largest corporations have the means and resources to embark on the development process regularly. Naturally, optimizing or improving any part of the pipeline has the potential to save billions of dollars in research development costs and hopefully save many lives.

Target-based screening
Within the preclinical part of the drug discovery process, up to millions of small molecule (compound) contenders, often spanning a large area of the chemical space, are synthesized, tested, and selected based

[1] JUMP-Cell Painting Consortium aims at creating a new data-driven approach to drug discovery based on cellular imaging and will create the necessary technical know-how and data to bring Cell Painting to bear on key bottlenecks in drug discovery research.

on their potential to cure the disease. Testing the efficacy and safety of so many compounds on organisms is unethical and impractical. Instead, each candidate small molecule is tested for its ability to physically bind a target that is hypothesized to be relevant to the disease. A *target* can refer to several types of cell components that the therapeutic acts on; typically targets are proteins such as receptors and enzymes, which are encoded by genes. The process of identifying compounds based on their strength of interaction with a particular target, as measured qualitatively or quantitatively using an assay, is called target-based screening. It is considered to be the workhorse of modern drug discovery [4]. Note that scientists carefully design each assay to fit many requirements such as low cost, speed, and specificity to the target disease. Therefore, developing the assay itself is a crucial part of the screening process.

1.2. Profiling

Phenotypic screening

The phenotypic approach poses an alternative to the above-described target-based method. Instead of testing the binding of each candidate compound to a target that is hypothesized to be relevant to the disease, in phenotypic screening the readout is chosen as a phenotype that is more directly related to the disease, such as the activation of a certain pathway in a reporter assay (where a candidate target is known) or the visible outgrowth of neurites in a microscopy assay. The advantage of the strategy is that in many cases, the phenotype is more directly related to the disease processes; the disadvantage is that setting up the cell-based assay is time-consuming and the assay itself is more expensive.

Introduction to profiling

Profiling - the characterization and quantification of multiple observable cellular traits - is emerging as a systematic, generalized assay that may accelerate various steps of drug discovery. When using a microscopy assay for profiling, the profiling process shares traits with the screening process, such as the model system with fluorescently stained cells. However, profiling follows a different philosophy. Instead of carefully handpicking a few features relevant to the disease, pathway, and assay, profiling captures a wide variety of features from a set of more general stains. This allows scientists to learn a lot more about the effects of a perturbation than a classical target-based screen would allow for, including unexpected effects. Additionally, a single profiling assay can be applied to various diseases or steps within the discovery process with no customization necessary, thus critically lowering costs and complexity. However, the vast feature space can also add unwanted noise and increase the difficulty of understanding and interpreting the output. Unsurprisingly, more customized assay readouts often exhibit a higher probability of finding a phenotype relevant to the disease.

Cell Painting

Advances across chemical, electronic, and microscopy technologies for imaging cells have led to a substantial increase in image-based profiling in terms of scale, resolution, and throughput [5]. Image-based profiling refers to the process of using automated microscopy and fluorescent staining to capture the morphology of a cell with as many features as possible. In the case of profiling, staining parameters are set to optimize costs and provide a wide range of unbiased cell organelles and components. One of the most common assays for image-based profiling is Cell Painting [1]. Cell Painting is the assay used in all baseline data for this project.

Challenges of profiling

Despite its advantages and promising future applications, especially when combined with advances

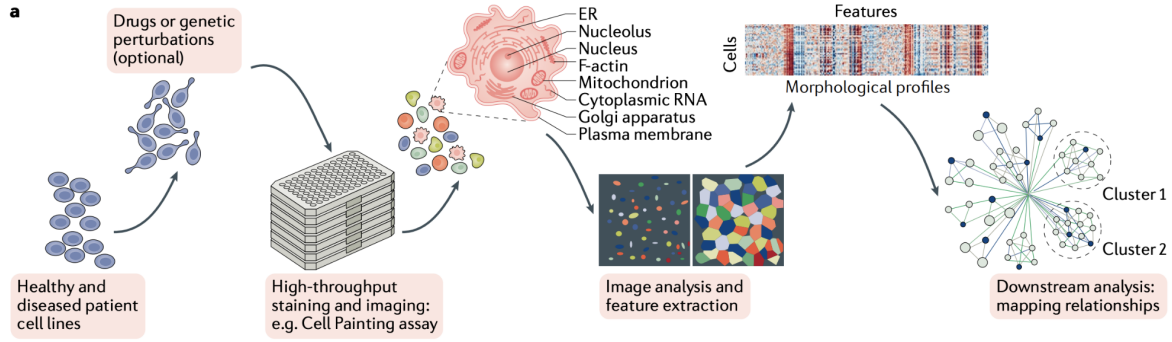

Figure 1.1.: *Profile extraction workflow.* Overview of the typical steps in the workflow for generating image-based profiles [1].

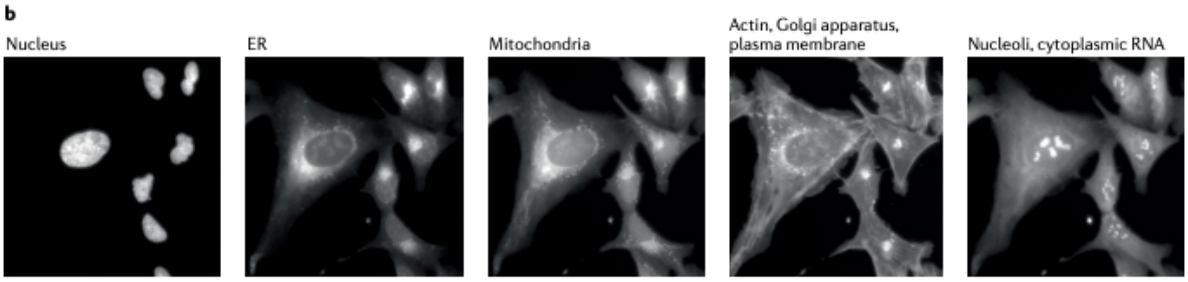

Figure 1.2.: *Cell Painting assay images.* Example images from the Cell Painting with its five channels [6]. ER, endoplasmic reticulum.

in computer vision and machine learning, profiling is not as widely employed by the pharmaceutical industry as the standard screening assays. Most commonly, it is used in stages downstream of screening, where profiling can be used as a second and unbiased form of validation. Here it can allow scientists to split compounds into groups of similar phenotypical and biological effects or even compare new candidate compounds to well-known compounds and thus hint or infer the mechanism of action (MOA).

1.2.1. Classical profiling approaches

Introduction to CellProfiler

The general workflow of classical image-based cell profiling consists of image acquisition and subsequent image processing and analysis [7]. Conventional image profiling approaches extract features at image level [8], [9] or at cell level [2], [10] via object segmentation. CellProfiler is the most commonly-used software for this purpose; it is open-source and can detect and segment cells, record their location, and extract features. Phenotypic profiles are measured during feature extraction, providing the raw data, often called single-cell profiles. The major types of features are shape, intensity, texture, and context (e.g., spatial relationships between cells) features [4].

Motivation for deep learning methods

While being successfully applied in various studies, including those done in the Carpenter-Singh Lab, these classical approaches have their limitations [4], [11]. The major weakness is often the specificity – the highly customized nature – of the profile extraction and downstream pipeline. Steps such as cell segmentation, selection of features, dimension reduction, batch correction, and many others require

knowledge of the specific assay used, the nature of hand-engineered features, or expected phenotypes. In other words, classic profile extraction is hard-coded to be optimized towards a particular assay, experimental setup, or downstream analysis metric. Applying a pipeline such as CellProfiler and other downstream processing to a different experimental design (e.g., different assay) may render the approach invalid or non-optimal. Fortunately, artificial neural networks, and in particular deep learning (DL) approaches, do not share this specificity due to the lack of a priori decision-making. Pawlowski et al. [12] nicely summarizes the arguments for DL as speed (faster feature extraction), autonomy (no need for human input to tweak parameters), and performance (higher scores on evaluation metrics). Furthermore, DL-based techniques may provide more avenues for addressing a significant roadblock in scaling up profiling experiments - the presence of technical artifacts in the signal precludes one's ability to combine datasets across different batches and experiments.

1.2.2. Deep learning-based profiling

Introduction to deep learning in cell imaging
Applying deep learning to image processing tasks has been a central branch of the deep learning community since ImageNet [13] and new architectures such as AlexNet [14] or EfficientNet [15]. Deep learning shows tremendous results on tasks such as the classification of radiographic images [16]. Hence, cellular images have also been the target of deep learning approaches. For instance, convolutional neural networks (CNNs) have been shown to perform cell segmentation and profile extraction with higher accuracy than traditional methods [4], [17], [18]. These successful applications feature the potential of image-based cell profiling coupled with deep learning methods for obtaining biological morphological activity for a broad set of applications. The adaptive nature of learning representations is what makes these approaches interesting. Manual features have limitations but data-driven features (i.e., learned features) are optimized to the specific variations of the data - in our case cell images. Building a system to automatically learn the representations of raw data is called representation learning and has been at the core of recent advances in computer vision [14], [19].

Technical artifact correction
Deep learning shows promise for tackling a central problem of profiling: technical artifacts. Technical artifacts, such as batch effects and plate layout effects, make cells from specific locations on the plate appear similar to each other and different from cells at a different location or from a different batch - even in the absence of true biological difference [20]. If technical artifacts are as strong or stronger than the biologically relevant features, extracting meaningful information from the data becomes a daunting or impossible task. Luckily, training on several batches and plates should allow the DL model to become more robust regarding technical artifacts. Recently, scientists have employed more advanced methods such as mixup [21], generative adversarial networks [20], and contrastive learning [22], [23] to further enhance model performance in the face of technical artifacts. I use image augmentation, L2 regularization, and label smoothing as tools to achieve this goal (see *Methods and Materials* chapter 3). Furthermore, a future neural network trained on cell images promises to extract normalized (centered and scaled) features that efficiently span the feature space, as normalization is inbuilt into a neural network via standard batch normalization [24].

Data integration
An additional advantage of DL applications is their modularity. First, tweaking hyperparameters or retraining a model on an improved dataset provide easily accessible ways to improve your profiling

pipeline. This would not be feasible with a classical method without introducing new or enhanced software that may be complex to add. Secondly, DL models are modular in regards to their input shape. For example, Becker et al. [25] showed how to merge chemical structures and phenotypical data into a machine learning model, thus improving their assay prediction.

Training models

Relevant work
In profiling, we can interpret the extraction of relevant features directly from the images' raw pixels as a computer vision task for deep learning models to solve. Pawlowski et al. [12] has successfully employed models trained on natural images (e.g., ImageNet) to cell images, while other authors have used transfer learning to combine pre-trained nets and (cell image) domain adaptation [26]. Esteva et al. [16] present an excellent example of successful transfer learning by reaching expert-level classification performance after fine-tuning an ImageNet pre-trained CNN to the domain of skin cancer diagnostics. Furthermore, Goldsborough et al. [27] used generative adversarial networks for learning profiles and showed them to outperform autoencoder-based approaches when evaluated for their mechanism of action (MOA) classification performance.

Motivating weakly-supervised classification
Like many other medical and biological datasets, we do not have extensive phenotypic ground truth annotations in my dataset [5]. However, one can expect that cells exposed to the same perturbations should look similar (similarity assumption) [28]. Therefore we choose to train deep CNNs using a weakly-supervised approach that applies the similarity assumption. Caicedo et al. [29] showed promising results by training a single-cell classification model (using the cells' perturbation as class labels) as an auxiliary task and then using the internal representation space of the resulting model to profile cell images. The resulting treatment-level profiles are evaluated on the basis of the similarity of compounds that share the same MOA. I employ MOA profile similarity as my central evaluation metric because it directly reports on the biology we care about in drug discovery. The weakly-supervised training approach was chosen because it yields promising results, is scalable, and is easily accessible through existing software (called DeepProfiler). My goal in this thesis is to develop a pipeline that can create image-based profiles of over two billion cells. I will build on Caicedo's work and perform weakly-supervised learning on millions of single-cell images. To the best of my knowledge, my thesis is the first to apply this methodology to a data set of over 18 million single-cells - the LINCS Cell Painting dataset.

1.2.3. Downstream Processing

After the feature extraction from classical or deep learning pipelines, further steps need to be taken to prepare the data for analysis. Operations include, but are not limited to, quality control and removing outliers, normalization, batch effect correction, dimensionality reduction, and aggregation. The review paper by Caicedo et al. [7] provides an excellent overview of best practices in cell profiling and downstream analysis. The *Methods and Materials* chapter 3 will detail the mathematical background of the methods.

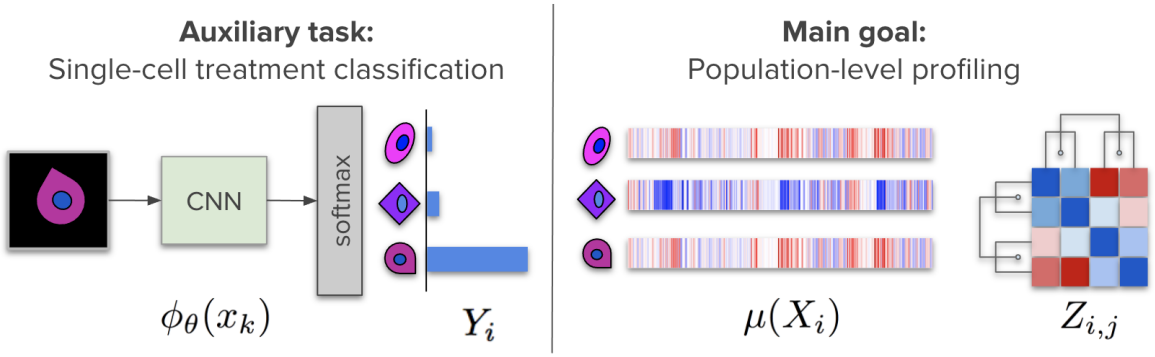

Figure 1.3.: *Overview of the training methodology.* A single-cell classification model is trained as an auxiliary task, while the main goal is to acquire good profiles from the model's representation space. Adapted from [29].

Common downstream methods:

- **Feature transformation** reduces differences in feature distributions or adjusts for population context.
- **Feature selection** and **dimensionality reduction** mitigate issues related to high dimensional spaces (curse of dimensionality) by, for example. reproducibility-based feature selection [30] or removing highly-correlated [31] features.
- **Normalization** brings features to the same scale and yields a common "origin" for profiles.
- **Aggregation** is used to create per-well or per-compound profiles.
- **Batch and plate-position effect detection** and correction can be done via median polish [32], detecting spatial patterns [33], and related methods.

1.3. Applications of Profiling

While specific assays are still the norm for many reasons, profiling has become the method of choice for a handful of companies. Most notably, Recursion, Janssen, and Insitro have successfully applied profile-based phenotypic screening with unbiased assays in areas that other large pharmaceutical companies have deprioritized [18]. Furthermore, Simm et al. [34] were able to use profiles from an old assay with a supervised machine learning approach to predict the activity of compounds in a different assay. They go as far as to suggest that high-content profiles can be used to "*[...] predict and replace customized biological assays*".

In the following paragraphs, I will present three common applications for profiling in the drug discovery process. For a more detailed description and other applications, the reader may refer to the review by Chandrasekaran et al. [4].

Profile-based phenotype discovery

The steps of profile-based phenotype discovery are as follows: first, the phenotypic differences (the screening objective) between the healthy and disease state must be found. This phenotypic difference is formalized as a vector in the profile space from the disease node to the healthy node. Next, pharmaceutical companies evaluate thousands or millions of compounds for their ability to reverse that phenotypic vector, i.e., to reverse the disease cell to the healthy cell. Traditionally, scientists have physically tested compounds on diseased cells and evaluated them for similarity of the resulting cells to healthy cells. Recently, profile databases have been used to look up compounds that move cells along the phenotype vector [35]. Ideally, the smaller and virtually selected set of compounds will exhibit the correct behavior on physical cells decreasing the costs of this process. Additionally, large datasets of image-based profiles can be queried for the phenotype vector providing insights into the compound structure and mechanistic pathway. Once completed in 2022, the JUMP dataset will represent the largest dataset of Cell Painting profiles to date, testing over 140,000 different compounds and genetic perturbations targeting $\sim$12,000 genes. JUMP aims to aid the profile-based phenotype discovery of the future.

Lead generation

Lead generation is the step in which pharmaceutical firms narrow down the hundreds of candidates coming out of screening to a few lead candidates. Here, compounds are modified and triaged to find the best attributes for the further process. While, again, specialized screening is the norm, unbiased profiles can provide new insights into the lead candidates: Profiles can, for example, easily find relationships among leads and cluster them into biological clusters and aid the process of identifying the mechanism of action [36]. Also, lead modifications that act on different biochemical pathways can be identified because image-based profiling covers such a broad spectrum of morphology.

MOA Identification

Identifying the MOA of a drug is beneficial because it gives biochemical insight, aids in optimizing leads, and increases the chances of clinical approval [37], among other advantages. This application of profiling data makes use of the assumption that compounds with similar MOAs generate similar profiles. When comparing a compound with an unknown target to an extensive database of annotated compounds, the MOAs of the nearest neighbors give a good indication of the compound's MOA. This so-called "guilt by association" method is not widely employed in industry because, although it has been effective in more than a decade of proof of principle studies [38], it appears to work for a relatively small proportion of MOA classes [4]. Nevertheless, I employed MOA profile similarity as my central evaluation metric, as this is one of the few applications in profiling with significant amounts of annotated ground truth, where correct answers for similarity are known. A similar clustering strategy can be applied by searching for the similarity of a compound's profile to those of genetic perturbations. Here, genes are knocked out (via CRISPR-Cas9, for example) or overexpressed and profiles are compared to the unknown-MOA compound. However, these nearest-neighbor-based approaches are considerably confounded by polypharmacology. Polypharmacology refers to the fact that most chemicals impact an array of proteins within a cell. Fortunately, recent work [34] has shown that deep learning models may be well suited to tackle this complex issue.

Other applications and future directions

Note that a wide range of successful applications of image-based profiling can be found outside of the scope of my thesis. These vary from image-based diagnostics, where disease cell profiles can predict patient outcome [39], to personalized therapies, where profiles of bacterial strains are used to measure the response to different antibiotics.

From the computational, biological, and pharmaceutical sides, the usage of profiling is increasing. While the deep-learning community continues to improve architectures and strategies, biologists and industry leaders will apply more and more machine learning to their processes [40]. We anticipate that deep learning's unique ability to learn feature representations is far beyond those predefined by extraction software and that this will drive a new era of discovery.

2. Contributions

This chapter will outline my contributions to the JUMP-Cell Painting project, the Carpenter-Singh lab, and the scientific community in general. Science is a group effort and especially in this project, my work was not solely focused on solving a well-defined and bounded research question but instead focused on exploring a potential path for others to follow in the future. This makes it harder to define my contributions clearly. Thus, I will use this brief chapter to detail my work in a centralized fashion.

2.1. Preparations for JUMP

My project's work was motivated by the JUMP-Cell Painting consortium's drive to produce the largest dataset of Cell Painting profiles worldwide. While I am not the first to apply deep learning to Cell Painting images, this project represents the most extensive dataset processed with the entire DL pipeline from cell images, via DeepProfiler, to evaluation metrics. In preparation for the JUMP, which will be available in early 2022, I used LINCS as a proxy dataset to mimic the challenges that analyzing hundreds of millions of cells will yield. I am confident that my contributions and learnings from LINCS will significantly improve the processing of JUMP data.

All practical learnings have been documented in several forms, including presentations to the JUMP consortium, my Github repository, a full-day class on DeepProfiler, and this thesis document. These resources combined will enable members of the Imaging platform and scientists from the JUMP partner institutions to readily build upon my work.

The final part of the *Discussion* chapter 5 summarizes the central recommendations for JUMP on profiling large amounts of data. There I detail my findings on training deep learning models and how this affects the feature space representation of cells. Furthermore, my improvement of DeepProfiler and Cytominer-eval and standardization of the profile extraction pipeline from CHTC scripts to Docker images will be helpful to the consortium and future researchers.

2.2. Cytominer-eval repository

My most significant contributions to the Cytominer-eval repository are the Enrichment and Hit@k metrics. In the former case, I built on Caicedo's ideas for implementing the *Enrichment* score, a global metric based on the odds ratio. *Enrichment* has been employed extensively throughout the project. The *Hit@k* metric was born through discussions in the lab and further developed by myself. I iteratively developed the calculations and representation and finally formalized the code into a function of Cytominer-eval. Furthermore, I fixed and updated the *Precision@K* and *Recall@K* metrics and added *Precision@R* as an

additional option to *Precision@K*.

Some smaller contributions of mine included optimizing the metrics such that the similarity matrix computes only once - solving a major computational bottleneck. I was involved in reviewing a handful of upgrades and code refactorizations that further expanded the use cases of Cytominer-eval. Finally, I am currently working with partner researchers on further optimizations and additions to incorporate new use cases.

2.3. LINCS Cell Painting repository

I contributed a major rewriting of the main README within the LINCS (Library of Integrated Network-Based Cellular Signatures) repository that describes the profiling pipeline and allows a new user to retrace the exact processing steps of profiles and consensus data. Additionally, I reworked the workflow image, which now gives a correct overview of the steps involved. Finally, I explicitly include the spherized profiles because my analyses showed that these outperformed all other normalization techniques.

2.4. Pycytominer

My primary contribution to the Pycytominer repository was the function *DeepProfilerProcessing*. This function reads the output features of DeepProfiler, including the metadata, aggregates, and saves the data in a Pycytominer and Cytominer-eval readable data frame format. While a version of this functionality existed as preliminary code, I refactored it into a more efficient and simpler version. It is now a crucial step of every DeepProfiler experiment and is performed after each profiling step. Moreover, I made a final update when DeepProfiler switched to TensorFlow 2.5 and changed the structure of feature outputs to compensate for larger amounts of data. Similar to the LINCS repo, I also updated the workflow image.

2.5. CP and DP downstream analysis

In my project, I compared CellProfiler and DeepProfiler profiles extensively with a range of evaluation metrics. In this thesis, I show which steps of the downstream processing pipeline improve metrics for CellProfiler and DeepProfiler data, respectively. I have developed a standard downstream pipeline for DP data and have written auxiliary functions that simplifies the Pycytominer and Cytominer-eval functions into a few lines of code.

2.6. DeepProfiler

Throughout the project, I was the main user and thus driver of improvements for DeepProfiler (DP). While I only wrote a few pull requests myself, I initiated and drove many issues and updates, which other contributors then implemented. All of the improvements have now been integrated in the official

0.3.0 release of DeepProfiler. The biggest change to DeepProfiler was the switch from Tensorflow 1.5 to 2.5. This update was necessary due to the new generation of NVIDIA GPUs, which run on CUDA 11. The Tensorflow update prompted us to perform significant adjustments to the training process and the model plugins.

Improvements to DP
Next, high overfitting results on my experiments motivated us to add image augmentations, L2 regularization, and label smoothing to the training routine (see *Methods and Materials* 3). Additionally, several changes were made to accommodate the increasing dataset size. The new DP version requires the user to export all cropped images before training, thus enabling users to apply any train-test-split and data subset strategy by manipulating the index file. Hence allowing users to define smaller training samples to decrease training time - an essential option while testing and optimizing hyperparameters. Furthermore, large numbers of small files in a folder created unwanted issues with file access and storage, prompting us to make the folder structure of images, crops, and feature output more granular. Besides the above-mentioned changes, I discovered several bugs while running my experiments. These were fixed, and the resulting program improved user-friendliness considerably.

Finally, my direct code contributions included the new folder structure for images, crops, and profiles and helper functions for checking input images, crops, and location files. These helper functions help ensure that all data is complete and correct and, most importantly, will not lead to a failure of the DP functions.

2.7. Server infrastructure

I investigated different EC2 instance types, server configurations, and AWS Images for hosting DeepProfiler experiments on AWS. This allows others to quickly start and run DP training and profiling for their own research, papers, and experiments. Being the first user within my lab to work on the CHTC servers, I tested and experimented with many infrastructural aspects, from the right server configurations to optimizing condor scripts for long training experiments.

Furthermore, I have been hosting all DP work on Docker containers that mimic different versions of DeepProfiler. I have created an extensive guide for DeepProfiler usage on CHTC which includes prewritten scripts. These allow new users to run the full process from sampling, training, profiling, and aggregation without much prior knowledge of DP.

Finally, I also started working on a distributed version of DeepProfiler, allowing us to run projects in parallel with an AWS fleet. This project was not completed, however, since the computing power of CHTC is sufficient for now.

2.8. Training models on cell images

As an alternative to using pre-trained CNNs, I spent a large portion of my effort on evaluating the infrastructure and resources needed to train neural networks with our weakly-supervised approach. This will allow the JUMP project to decide if and how to train models on the JUMP data to improve the profile

feature space and subsequent research. The Experiments and Results chapter details the exact work done in this regard, hence, I will only give a short summary here.

I extensively examined the potential of deep learning profile extraction strategies (pre-trained models or model's trained under weakly-supervised classification) - and compared these to existing methods (CellProfiler). In over 50 different deep learning experiments, I investigated the effect of several training hyperparameters on the performance of their resulting profiles. Optimal values or value ranges were found for the learning rate, batch size, label smoothing parameter, epoch number, augmentation, and for the number of cells that we train on. By utilizing different subsets of data, I was further able to show that trained models improve profile extraction for the type of cells that the model was trained on, proving that I am indeed learning representations. On the LINCS dataset, I matched the performance of pre-trained models with my own trained models.

Moreover, I formalized the optimal pipeline for profiles from the trained and pre-trained strategy. I found that spherizing creates an orthogonal feature space, diminishes batch effects, and generates the best-performing profiles.

Overall, I have prepared a clear strategy for the JUMP-Cell Painting Consortium to follow in the upcoming months and have recommended an extensive list of further research questions to investigate. I hope that my work will be instrumental in profiling the two billion cells of the JUMP data set in the months to come.

3. Methods and Materials

This chapter will introduce the materials and methods that led to the final results in the experimental chapter. I will describe the LINCS data – the dataset on which I performed all my experiments. I will then describe the software libraries used for training (DeepProfiler), downstream processing (Pycytominer), and on evaluation (Cytominer-eval).

3.1. The LINCS Cell Painting Dataset

Motivation
To prepare a computational pipeline that would also be applicable for the very large JUMP Cell Painting data (referred to as "JUMP" hereafter) in 2022, we decided to run all experiments on the LINCS Cell Painting dataset [41], [42], [43], which contains more compounds than most other available datasets and shares many similarities with the JUMP data. Using LINCS enables me to find the best possible pipeline to profile JUMP data next year. The repository contains two sets of data: the "LINCS Pilot 1" and the "LKCP" dataset. I will solely focus on the former set and simply refer to it as "the dataset" or "LINCS". The repository provides information, such as links to other collaborators and a list of compounds and plate maps used in the database.

Details of the data
The dataset contains approximately 100 million A549 cells (lung cancer) perturbed with 1,571 different compounds across six doses and five technical replicates. Technical replicates refer to a group of wells perturbed in the same manner – the same compound at the same concentration, with all other experimental parameters being identical. Technical replicates must not be confused with what is later referred to as the "same-MOA compound matches". LINCS comprises 136 384-well plates [44] (52,224 wells in total) with 28 different plate maps. Nine "fields of view" or "sites" were imaged from each well. Each image has five channels (the five channels captured in the standard Cell Painting protocol [1]. The experiments were performed on five different days, marked with a batch number from 1 to 5. The Appendix 6 and my repository contain sample images of cells and notebooks with a detailed analysis of this dataset.

Polypharmacology
An important note about how I handled polypharmacology: if a compound has multiple MOA annotations, these are concatenated and become its own unique MOA. Thus a compound with *MOA_a & MOA_b* does not MOA match with a compound with (only) *MOA_a*. This simplification decreases our performance scores but does not affect our ability to compare different pipelines and methods. The number of compounds tagged with these "composite" MOAs are listed in Table A.1 in the *Appendix*.

The LINCS repository also includes a range of profiles acquired through the classical pipeline of CellProfiler and Pycytominer. The reader will benefit from reading at least the profiles/README because

I use some of these profiles for baseline comparison, and many of the LINCS pipeline steps overlap with my pipeline.

3.2. DeepProfiler

We chose DeepProfiler to be the primary tool for running deep learning experiments due to its ability to process Cell Painting images and run the weakly supervised classification experiments.

Introduction

DeepProfiler (DP) is a set of tools written by Caicedo and colleagues. This program allows machine learning scientists to train deep learning models and infer profiles from Cell Painting images. The DeepProfiler repository is an active codebase and has developed considerably over the last few months. Many of these improvements arose from its application in my thesis and the accompanying requirements. Thus, the features and methods described on these pages may be quickly outdated, so I refer any interested readers to the Github pages for the most recent documentation.

The following sections detail the essential functions of DeepProfiler: exporting single cells, training a model, and inferring the cell profiles. In my thesis, I will focus solely on the theoretical aspects of DeepProfiler; however, I have accumulated a long list of practical tips and information for working with DP.

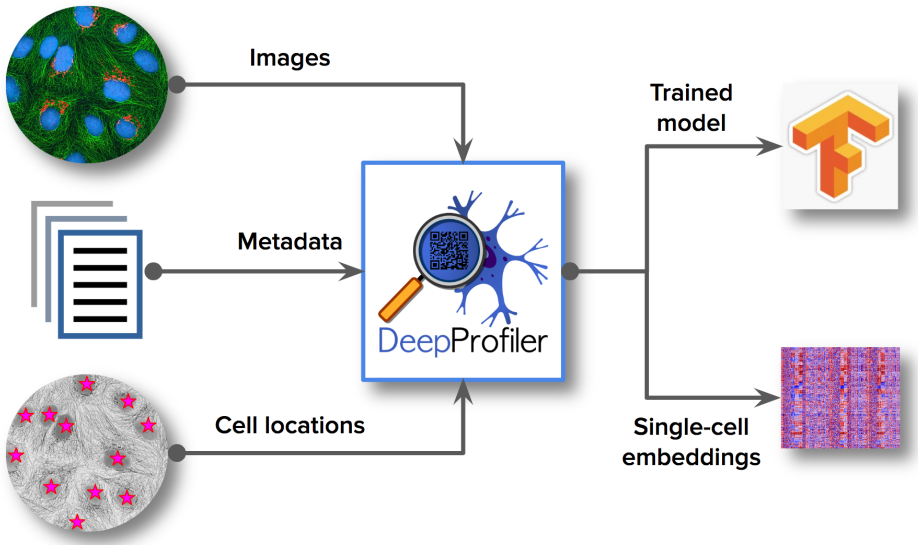

Figure 3.1.: Depiction of the input and output of DeepProfiler.

3.2.1. Exporting single cells

Training requires random access to single-cell crops. Due to both computing and space constraints, cropping images during runtime or storing all cells in RAM is unattainable. Thus, the preferred solution in DeepProfiler is to pre-sample (export) all input images into [128 by 5x128] pixel size crops with all five channels conveniently concatenated into one file. DP loads a random subsample of these millions

of single-cell images into cache at training runtime, which constitutes a training batch. Due to time constraints, it can be advantageous to use only a portion of the entire dataset for training by including fewer compounds or fewer cells from each well. This subsampling of training data is discussed in detail in the *Experiments and Results* chapter 4, and an example of exported single cells can be found in the Appendix 6.

3.2.2. Train

Training models on the auxiliary task of compound classification and optimizing the evaluation metrics is the focus of the *Experiments and Results* chapter. Multiple parameters can influence the training model, such as architecture, train/test split, epoch number, batch size, learning rate, image augmentation, label smoothing, and many others. I used the ImageNet pre-trained EfficientNet architecture [15] as weight initialization for most experiments and trained the classification model for 15-30 epochs until the CNN converged. The trained model was then used to infer profiles and calculate evaluation metric performance.

The following sections detail relevant parameters of the training configuration. Augmentation, L2 regularization, and label smoothing are operations aimed at regularization. Regularization refers to a suite of techniques that improve a deep learning models' ability to generalize and perform better on unseen data by keeping overfitting in check.

Augmentation

Purpose of regularization
Powerful neural nets are heavily reliant on big data to avoid overfitting. Data augmentation refers to the process of enhancing the quantity and quality of a dataset by applying transformations to the color space, geometry, style, or several other attributes of the images. We hypothesize that these augmentation operations will impede the model from overfitting and increase its robustness towards technical artifacts. If different wells have different illumination values due to their position on the plate, we hope to obscure these differences with random illumination increases and decreases.

DeepProfiler augments single-cell images during training. The augmentation operations include random crops, random flips, and random illumination adjustments. First, we crop the image (which itself is a single-cell crop) with a 50% probability with a crop size of 80-100%. Next, random horizontal flips and 90-degree rotations are applied, and brightness and contrast variations are performed. First, Image brightness is randomly increased or decreased by 10%, and next, the contrast is randomly varied between 80% and 120%. Users can turn the augmentation mode on and off by adding the correct line to the config file.

L2 regularization

L2 regularization is the most common type of regularization. It updates the general cost function by adding a penalty based on the weight parameters. The implicit assumption here is that smaller weights lead to a simpler model. However, if the penalty is too large, the model will underfit. The penalty term added to the loss function is determined by the weight decay parameter λ:

$$l = l_c + \lambda \sum_{j=1}^{p} \beta_j^2, \tag{3.1}$$

where l_c is the standard classification loss, p is the number of layers, and β_j^2 is the square of the weights in the j^{th} layer. In my experiments, I use a weight decay parameter of 0.0001.

Label smoothing

When performing image classification tasks, we typically think of labels as hard assignments and use one-hot encoded vectors as the truth label. However, there is mathematical and empirical evidence that this is not the best option, and we do not want our model to become too confident in its predictions [45]. Thus label smoothing can be applied to impede overfitting and overconfidence in the model. In a sense, label smoothing is the functional equivalent of data augmentations since it softens and broadens the space that the model is learning. The function computes smooth labels by replacing one-hot encoded vectors with a combination of one-hot encoding and the uniform distribution:

$$v_s = v_o \times (1 - l) + \frac{l}{N_c}, \tag{3.2}$$

where l is the label smoothing variable, v_o the old onehot label, v_s the new and smoothed onehot label, and N_c the number of classes.

Learning Rate

The learning rate hyperparameter controls how much the model weights change in response to the loss function error. Choosing the correct learning rate is crucial but also challenging, as a small value can result in very long training times. At the same time, too large values result in unstable gradient descent and potential non-convergence.

Types of learning schedules
Three learning rate strategies are supported: 1) The rate can be constant throughout all epochs. 2) The step schedule changes the learning rate to a desired value at specific epochs. 3) Cosine decay increments the learning rate from zero to the initial learning rate using a linear function during the first five epochs. After that, every epoch reduces the learning rate according to the cosine decay. Cosine decay is applied because it is often helpful to lower the learning rate as gradient descent approaches a minimum.

3.2.3. Profile

The profile method allows users to infer single-cell embedding (profiles) with the site image, the metadata, cell locations, and a trained neural network model. The model can be pre-trained, e.g., an EfficientNet trained on ImageNet data, or DeepProfiler itself can train it. The config files further allow defining the feature layer which is extracted. Typically, we will extract the second-to-last feature level for the profiles, but one may choose to output the softmax classification level instead.

3.3. Pycytominer

Pycytominer [42] is a collection of tools that allows processing profiles from CellProfiler, DeepProfiler, and other profile extraction tools. Both the LINCS repository and my repository extensively make use of Pycytominer. An API allows for easy access to five different steps (aggregate, annotate, normalize, feature

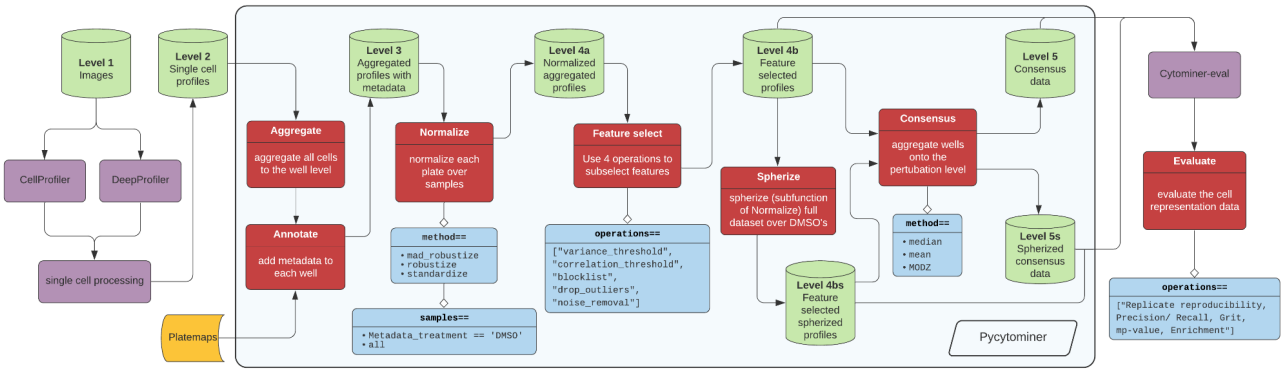

Figure 3.2.: Workflow image shows the most often used pipeline for downstream processing data. On the left, we start with single-cell profiles and output Level 5 compound data used to compute evaluation metrics.

select, consensus) (see 3.2), which process data from single-cell profiles (Level 2) to consensus signatures (Level 5). After the application of these operations, users can apply Cytominer-eval to compute evaluation metrics.

3.3.1. Pycytominer functions

The following subsections describe the five steps of a typical data processing workflow that's based on Pycytominer, described in order of their typical execution.

Aggregation

In the classical CP workflow, the aggregation method is used first to aggregate single-cell data (Level 2) onto well-level data (Level 3) and then later for the creation of the consensus profiles (Level 5). From Level 2 to 3, the median of all the single-cell profiles within a well is aggregated to one profile per well. From Level 4 to 5, the five technical replicates of each well are combined to a compound profile.

Annotate

The annotation function adds relevant metadata to the profiles. This is needed since the aggregation function does not incorporate metadata other than the information available in the experimental structure itself, which is typically just the plate id and well position.

Normalize

Normalization is the most crucial part of the process, since it is our main tool for alleviating batch effects from the profiles. The methods of normalization available are *Standardize*, *Robustize*, *Mad Mobustize*, and *Spherize*. I describe the mathematical detail of each operation in the section below.

Standardize

The *standardize* function simply transforms every feature by removing the mean and scaling to unit variance. Scaling and normalization can be performed over the entire plate or the negative controls (DMSO).

$$z = \frac{(x - \mu)}{\sigma}, \tag{3.3}$$

where x is the feature value, μ is the mean, and σ is the standard deviation of that feature. As noted, μ and σ are computed over the whole plate or the 24 DMSO wells on each plate. All other methods are also calculated per plate except for spherizing, which is a global operation.

Robustize

Robustize is very similar to Standardize but is more robust to outliers. It scales the data according to the interquartile range, i.e., the range between the first and the third quartile:

$$z = \frac{(x - \mu)}{Q_3 - Q_1}, \tag{3.4}$$

where Q_1 and Q_3 are the first and third quartiles, respectively.

Mad Robustize

Mad Robustize normalizes with respect to the median absolute deviation $\tilde{X} = \text{median}(X)$:

$$MAD = median(|X_i - \tilde{X}|) \tag{3.5}$$

and

$$z = \frac{(x - \mu)}{MAD}, \tag{3.6}$$

Of these three normalization operations, Mad Robustize is most commonly used on CellProfiler profiles, as seen in the LINCS repository.

Spherizing

Motivation behind spherizing

The idea of spherizing profiles using the distribution of negative controls was first introduced by Ando et al. [46] under *Typical Variation Normalization*. They report improving their ability to classify cell images into the correct perturbation and making their *"[...] results more robust to nuisance variation such as batch effect."* by, essentially, applying the spherizing operation. In short, the operation projects the data into the principal components of the negative controls in the experiment and then scales each axis by the variation of the negative controls in that axis such that the negative controls have unit variance in each component. The same transformation is then applied to the rest of the data. The normalization amplifies features with low DMSO variation and weakens those features with a high variation among the negative controls. Since DMSO cells should look very similar regardless of their well position, plate, and batch, those features with high variation are likely related to unwanted technical artifacts. In our experience, spherizing appears to be the best available method for amplifying phenotypic signals and reducing technical artifacts. Unlike the other normalization operations, we perform spherizing on the full dataset (all plates at once). Our custom spherize function transforms data such that negative controls form an identity covariance matrix after the transformation.

Mathematical description of spherizing
Firstly, let Σ be the covariance matrix of the data $X^{n\times d}$ with n data point and d features. Then $\Sigma = \frac{X^T X}{n}$ has eigenvectors in columns of U and eigenvalues in the diagonal of Δ, such that $C = U\Delta U^T$ is the eigendecomposition of Σ. Matrix U^T then maps the original features onto the principal components such that every component will have a variance given by the corresponding eigenvalue. Thus, dividing by the root of eigenvalues will create the identity covariance matrix $x \to \Delta^{-1/2}U^T x$. This is called the "normal" PCA sphering. However, the operation is not unique since the data will stay sphered after every rotation. The special case, ZCA spherizing, adds the rotation U such that $x \to U\Delta^{-1/2}U^T x$. The defining property of the ZCA transformation, in comparisson to the normal PCA transformation, is that it results in a state that has the least squares distance to the original data. Kessy et al. [47] describe more mathematical details and alternative spherizing approaches.

Feature selection

The feature select method can apply several modes: *variance threshold*, *correlation threshold*, *drop na columns*, *drop outliers*, *blocklist*, and *noiseremoval*. The Carpenter-Singh lab developed these functions to drop redundant and invariant features and clean the data from missing information.

- *Variance threshold* excludes features that have a low variance from the data frame. This function is also essential if later steps include a PCA or eigenvalue decomposition since very low variance features lead to a division-by-zero.
- *Correlation threshold* excludes features such that no two features correlate more than a specified threshold.
- *Drop na columns* excludes those profiles that hold some or too many NA values.
- *blocklist* simply excludes predefined features from downstream analysis.
- *Drop outlier features* exclude a feature if its minimum or maximum absolute value is greater than the threshold.
- *Noise removal* excludes features with excessive standard deviation within the same perturbation group.

These feature selection methods are crucial for CellProfiler profiles since features commonly correlate or have no measurable variance. I will show that for DeepProfiler data, feature selection becomes insignificant.

Consensus

The consensus function is another step of aggregation and completes the pipeline. We aggregate the normalized and feature selected data (Level 4b) via the Median or MODZ functions to consensus data (Level 5). In other words, all five technical replicates are combined to one profile representing one perturbation with a given dose.

DeepProfiler Processing

Since DeepProfiler outputs single-cell profiles in a different format than the output of CellProfiler, I have built DeepProfiler Processing, allowing users to aggregate DeepProfiler profiles with a mean or median operation onto the site, well, or plate level. This helper function demonstrates how to use *Pycytominer.DeepProfiler Processing*.

3.4. Cytominer Eval

Introduction to eval
The Cytominer-eval repository contains six functions that calculate different quality metrics for perturbation profiling experiments: *Precision@K* and *Recall@K*, *Enrichment*, *Hit@k*, *Replicate reproducibility*, *MP-value*, and *Grit*. The first four metrics will be employed in my thesis as evaluation methods. I will refer to a DeepProfiler model (with its corresponding downstream pipeline) that scores high on all four evaluation metrics as a high-performing model. I have chosen these metrics based on their ability to measure usefulness to the pharmaceutical process. While *Precision@K* and *Recall@K* are established methods, *Enrichment* is less frequently used, *Hit@K* is introduced in this thesis. In the following subsections, I detail the mathematical background and coding implementation of each metric.

Similarity-melted data frame

The similarity matrix
The similarity matrix is required to compute all three evaluation metrics. The matrix contains all connections of the fully connected graph of all samples given as input to Cytominer-eval. The similarity matrix is closely related to the distance matrix, but the Pearson Correlation Coefficient (ρ) is calculated (instead of using other similarity metrics, like cosine similarity or Euclidean distances). The (ρ) measures the covariance of the two variables divided by the product of their standard deviations:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{3.7}$$

where ρ is the correlation coefficient between the two samples X and Y, cov is the covariance, and σ is the standard deviation of X and Y.

Adding replicate labels
Additionally, we need to add truth labels to each connection of nodes. Replicate groups determine which vertices will be labeled as 'true'. If *replicate groups* are set to *MOA*, then all pairs of samples that share the same MOA will have their connection labeled as 'true'. If the replicate groups are *sample ID* and *dose*, then only pairs that are technical replicates will be marked 'true'.

A different way to think about the melted similarity data frame is to imagine it representing a fully connected network of nodes (samples) and vertices (similarity value). Each vertex is a row in the similarity melted data frame, and the relevant columns are Node A, Node B, Similarity score, and group replicates. Note that this representation is bidirectional. For the experiments in this thesis, the input to

the Cytominer-eval function is almost always a Level 5 (consensus level) profile, and the replicate group is the MOA. If compounds A and B share the same MOA, I will refer to A and B as "MOA matches" of each other. Based on this melted similarity matrix, all the following scores can be calculated.

3.4.1. Precision and Recall

In the field of statistics, the term "precision" is overloaded with different meanings across information retrieval and statistics. Average Precision (AP), Precision@k, and Precision@R are three common uses of the term. The following section will briefly detail these three versions and clarify which is present in the *precision_recall* function of Cytominer-eval.

Mathematical description of average Precision and Recall

Precision and recall
Arguably, the most popular evaluation metric used for classification is precision and recall. Used in statistics, Data Science, and Machine Learning, the precision of a classifier for a given class is calculated as the ratio of true positive (TP) and the total number of predicted positives. The formula is simple:

$$Precision = \frac{TP}{TP + FP} \tag{3.8}$$

where FP are the false positive predictions. Equally simple is the recall, i.e., the true positive rate (sometimes called sensitivity) of a class in the classification:

$$Recall = \frac{TP}{TP + FN} \tag{3.9}$$

where $TP + FN$ are the total number of positives.

Examples
Note that, by definition, there lies a trade-off between precision and recall performance. For precision to be high, we would want to decrease the number of false positives, but this will decrease our recall and vice versa. Real-world requirements often determine this tradeoff decision (e.g., Email spam filter vs. medical decision model). In our case, the tradeoff parameter is k, the number of top connections we include in the prediction set. So given a class A, we "predict" that the k nearest neighbors will share the same MOA as class A. If k=10 and TP=3, precision will be 0.3, and recall will be three divided by the total number of MOA matches (the number of other compounds with the same MOA).

This sample precision is calculated for all samples and aggregated to a single value to represent the total precision:

$$precision(k) = \frac{\sum_s^N precision(s,k)}{N} \tag{3.10}$$

where s is a sample, $precision(s,k)$ is the precision at k for a sample s, and N is the number of samples. This definition is our first meaning of precision and also the definition that will be used for the evaluation of profiles in the *Experiments and Results* chapter 4. To avoid confusion, I will henceforth refer to this value simply as *Precision@K*.

Challenges with fixed k

When class sizes vary greatly, *Precision@K* can display unwanted behavior. Assume calculating precision at 10 for a compound with MOA class size 2 (the compound only has one MOA match). Then Precision@10 will be either 0 or 0.1, and the Recall@10 will be either 0 or 1. If on the other extreme, we are calculating the Precision@5 for a compound with 20 MOA matches, then the precision will likely be very high, but Recall@5 will be capped at 0.25. The LINCS data does indeed show this problematic behavior.

Precision@R

A partial solution to this problem is to calculate *Precision@R* instead. *Precision@R* is *Precision@K* where r is the class prevalence (or MOA class size):

$$precision(r) = \frac{\sum_s^N precision(r(s))}{N} \tag{3.11}$$

where $r(s)$ is the class size of sample s. I will report this metric in the *Experiments and Results* chapter and test its alignment with other metrics.

Average Precision

Interestingly, we can borrow metrics from the field of information retrieval because searching for MOAs is very similar to searching for the same-MOA compound [48]. In information retrieval, the term average precision often changes its meaning. When predictive systems return ranked sequences of items (e.g., documents or classes), it is essential to consider the order of these returned items. A more detailed description of our classification can be obtained by computing precision $P(k)$ and recall $R(k)$ at every k and plotting the precision-recall curve, $P(R(k))$ is. We call the average value of $P(R(k))$ from $R \in [0, 1]$ the Average Precision (AP) [49]:

$$AP = \int_0^1 p(r)dr = \sum_{m=1}^{k} P(m)\Delta r(m) \tag{3.12}$$

where m is the rank in the k long list of nearest neighbors and $\Delta r(m)$ is the change in recall from $k-1$ to k. Hence, it is clear why Average Precision is also referred to as Area Under the Precision-Recall Curve (AUPRC). The above sum is equivalent to

$$AP = \frac{\sum_{m=1}^{k} P(m) \times rel(m)}{N_c} \tag{3.13}$$

where N_c is the number of replicate classes and $rel(m)$ is the indicator function that equals one if the item at rank m is a same-MOA compound and zero otherwise. The mean Average Precision (mAP) over all samples gives us a single value describing the data set in terms of MOA prediction by nearest neighbors.

$$mAP = \frac{1}{N} \sum_{s=1}^{N} AP_s \tag{3.14}$$

Implementation in Cytominer-eval

Of the three different options for Precision, I have chosen to implement *Precision@K* and *Precision@R* in Cytominer-eval and use them throughout the *Experiments and Results* chapter 4 as an indicator for performance. To be clear, I report *Precision@K* using macro-averaging, i.e., I compute values per compound and average across all compounds. I did not implement Average Precision (AP), as it tends to be empirically very similar to *Precision@R* [48].

3.4.2. Enrichment

The *Enrichment* score [50] approximates how likely the same-MOA pairs are in highly (Pearson) correlated versus less correlated sample pairs. *Enrichment* is based on the odds ratio in a one-sided Fisher's exact test. The odds ratio is calculated by assuming the 2 x 2 frequency table 3.1 with the predictions as rows and the labels (MOA matches) as columns. We predict a connection to be true if the similarity is above a certain percent threshold t. The odds ratio is the ratio of values in the first row divided by the second row. In the *Experiments and Results* chapter, I often calculate the odds ratio for the top percentile, e.g., from 99.5 % - 97%. Note that the percentile I report (99.5%) corresponds to the same percentile of 0.5% used in Rohban et al. [50]; our sorting order when reporting the rank is opposite, but everything else remains the same.

$$oddsratio = \frac{(a/b)}{(c/d)} \tag{3.15}$$

	MOA replica	non-MOA replica
highly correlated	a	b
weakly correlated	c	d

Table 3.1.: Figure of the 2 x2 frequency table for calculating the odds ratio.

This evaluation metric makes no assumptions about MOA class sizes and does not suffer the challenges of *Precision@k* with a fixed k. *Enrichment* is a global metric because it computes the odds ratio on the fully-connected graph of compounds and similarity connections instead of working at the compound level.

3.4.3. Hit@k

I developed *Hit@k* intending to create a metric that combines the granularity of *Precision@K* and creates a global value, like *Enrichment*. My goal was to provide a metric that would be easily understandable and with an intuitive visualization for biologists. *Hit@k* is the closest to the application of MOA search.

Calculation of Hit@K
Hit@k is very similar to *Precision@K* and *Recall@K* in that it groups connections by compound and orders all nearest neighbors by their similarity. Then, instead of calculating the precision, the function simply counts the rank of all correct same-MOA matches. The function name refers to the correct same-MOA

compound neighbor (hit) at the kˆth position. It concatenates all hits into one long list called *hits list*. For a random distribution of points, we would assume the distribution of hits to be uniform between zero and the number of samples. However, if compounds are close to their same-MOA matches, the hits list will be heavily skewed towards the lower ranks. The simplest method to plot and compare this metric is to create a histogram from the hits list. This histogram plot gives direct insight into how many correct same-MOA matches are within the first nearest neighbors - this is the value of the first histogram bar (366 in Figure 3.3).

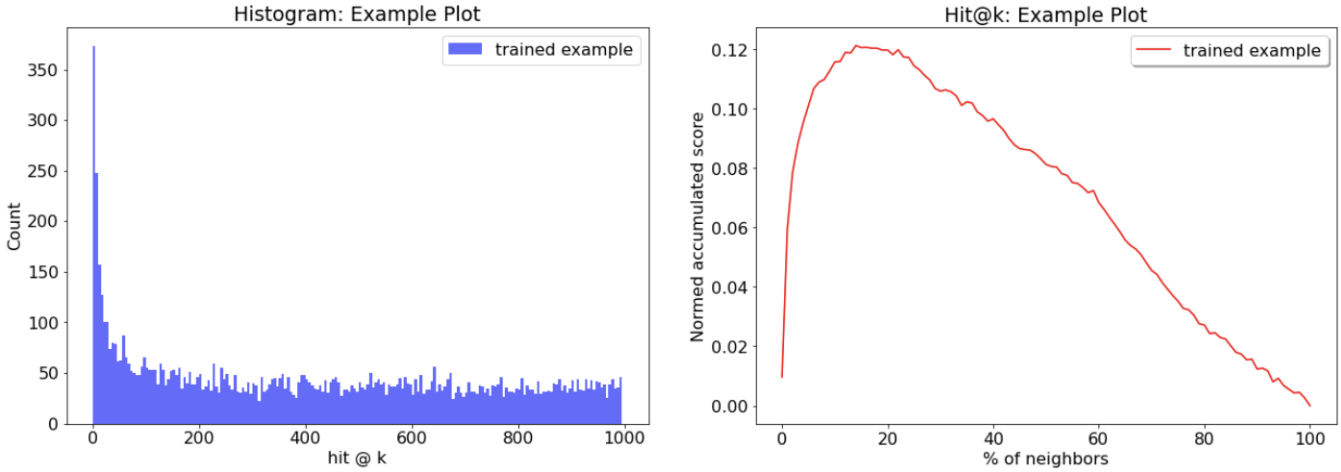

Figure 3.3.: Left: The histogram plot of a hits list with bins of width five. Over 350 hits were found in within the five nearest neighbor. Right: Normed accumulated scores allow for better comparison of several profiles

However, histogram plots are hard to compare to one another, so I decided to display the accumulated scores of the histogram. The normalized accumulated scores of the histogram are calculated by adding the value of each bin and subtracting the expected value per bin. The expected value per bin is simply the number of all hits divided by the number of bins. In other words, the normalized accumulated scores are the cumulative density function (CDF) of the histogram, subtracted by the expected values. Figure 3.3 shows the histogram and the respective normalized accumulated score plot. Note that the accumulated scores are binned into percentages so that it does not matter how large the number of compounds is.

3.5. High-performance computing resources

Part of the goal of my project is to give recommendations regarding the computing and time resources needed to run the profiling pipelines. Hence, this section will give a brief overview of the computing resources tested and used.

AWS

The elastic computing (EC2) service from Amazon allows quick access to GPU resources. I used two different instances:

- P2: The cheapest GPU on EC2 has an NVIDIA Tesla K80
- P3: Next EC2 generation with an NVIDIA Tesla V100

CHTC

The majority of my experiments ran on the Center for High Throughput Computing servers. I was able to use a large server with access to 4 NVIDIA Ampere 100 SXM4 GPUs. These allowed me to run training, profiling, and exporting in reasonable time formats.

4. Experiments and Results

Overview of this chapter
This chapter covers the relevant experimental aspects of my thesis. First, I describe the relevant attributes of the subset of the LINCS data used for the experiment. Next, the *metric comparison* section introduces the evaluation metric plots and the *downstream pipeline* section outlines my decision on what data processing methods from Pycytominer to apply before evaluating the profiles. I then show the performance of the CellProfiler profiles and pre-trained models, and finally, introduce trained models and detail how training hyperparameters affect the performance.

A large number of training experiment with different hyperparameters and training data is performed and presented in this chapter. Describing the details of each experiment within the text or figure description is too cumbersome. Hence, I include a list of experiments with unique ID labels in the *Appendix* 6, where I detail the parameters of each experiment, including a link to the raw data.

This chapter solely focuses on the experimental design, data and results keeping the reading time of this chapter to a minimum. A discussion, conclusion and recommendation for further research follows in the next chapter. In order to allow for easy navigation between the experiment and discussion, I will include hyperlinks at the end of each section that point to the corresponding discussion. Moreover, the sections of both chapters are identical, e.g., chapter 4.1 corresponds to 5.1

4.1. Experimental design

4.1.1. Data selection

I do not require the full LINCS dataset for my experiment and using a subset increases the speed of my experiment. Thus, I will briefly point out some important details about the subset used.

Motivation to select a subset
The evaluation metrics all require the number of compounds in an MOA to be at least two (MOA class size of at least two). A compound that does not have other compounds annotated with the same MOA cannot have a precision value and does not affect *Enrichment* and *Hit@k*. Additionally, there is no use for compounds with unknown MOA labels, because I can not apply any metrics. Finally, I used only a single dose per compound to make experimental iterations practical. I picked the 10uM dose point because we found that this dose produces the strongest phenotypes across compounds without reducing the ability to group MOAs [42]. These three requirements lead to a subset of the LINCS data in which I 1) only keep the highest dose of each perturbation, 2) delete all compounds that have unknown MOA labels, and 3) drop all compounds with MOA sizes smaller than two. All further processing and experiment are based on this subset of data. This subset process is documented in my repository.

Reiterate important LINCS details
The resulting (sub-selected) dataset comprises 8,818 wells spanning over 136 plates, five different batches, and 18.2 million cells. Each of the 1,144 perturbations (compounds) have five technical replicates - each on a separate plate but in the same position on the plate (same well position). Four of these technical replicate wells were produced in the same batch. This is highly relevant for training and technical artifacts, because I expect cells in the same well and same batch to have similar technical artifacts, but the corresponding wells from a different batch may look very different from one another. Each of the 136 plates has 24 negative controls (DMSO) distributed across the plate (3264 DMSOs in total). The 1,144 compounds can be categorized into 235 MOAs, and every MOA has at least two compounds (MOA class size > 1). Some MOAs, such as *phosphodiesterase inhibitor,* have an MOA class size of 35, while most other MOAs are only found for two compounds.

4.1.2. Profile extraction for pre-trained and trained models

Pre-trained nets and pipeline
DeepProfiler's profiling function takes images and single-cell locations and runs cropped cell images through a pre-trained convolutional network. Both EfficientNetB0 and ResNet50 models have been trained on the ImageNet dataset and can be downloaded through the Tensorflow API. Because the models require RGB channel (3 x N) input vectors, each Cell Painting channel is replicated three times to fit the model input. Hence, the models infer a *N_out*-dimensional feature space for each channel, and DeepProfiler then concatenates the output vectors. This results in (5 x 1280) 6400 and (5 x 2048) 10,240 dimensional profile vectors, for EfficientNet and ResNet respectively. Single-cell profiles are then aggregated to well-level data with the Pycytominer.DeepProfiler_processing function. I refer to this data as pre-trained profiles.

To acquire so-called *trained profiles*, I train an EfficientNet or ResNet model on single-cell crops (see *Exporting single cells* 3.2.1) with all five channels contained in a single image (Tensorflow has a variable training input shape). Thus, the size of the output vector is only 1280 (2048 for ResNet50) - the default size of the pooling layer of the network. After training, I followed the same steps as in the case of the pre-trained nets.

Link to Discussion 5.1

4.2. Metric comparison

Using several metrics
Before applying the evaluation metrics to data and looking for the best-performing representations, I want to compare all metrics to each other on three profiles from the three different profiling strategies: 1) an example trained profile (ID: 813), 2) the pre-trained baseline (ID: 200), 3) and the CellProfiler baseline (CP baseline, ID: 100). I have chosen a handful of metrics that measure different aspects of the data, allowing for a broader understanding of the results and higher confidence that the learned feature space is better suited for its applications. On the downside, the metrics can disagree and complicate the interpretation of the results. Figures 4.1 and 4.2 show three typical plots (including *Enrichment*, *Precision@K*, *Recall@K* and *Hit@K*) and one table (Table 4.1) for the *Precision@R* values. I refer to metrics as agreeing with one another if they agree on the best performing strategy. As a brief reminder, *Precision@K*, *Precision@R* and

Recall@K are calculated using macro-averaging, i.e., they compute precision values per compound and then average across all compounds.

Comparing metrics

The pre-trained profiling strategy - which I also refer to as *pre-trained profiles or PT baseline* - appears to be the best performing over all five metrics. While the trained profiling strategy (or *trained profiles*) shows higher *Enrichment* and *Precision@K* scores it underperforms in all others scores when compared to CP baseline. This example shows the advantage of employing different metrics, because the *Enrichment* metrics alone would not have revealed the full nature of the performance of these profiling strategies. For simplicity, I will say: "*the trained model outperforms the CP baseline*", even though the models themselves do not outperform but their resulting profiles do.

Overall, *Enrichment*, *Recall@K* and both *Precision@K* and *Precision@R* scores agree on the best performing data most of the time. In the *Appendix* 6, I show how *Precision@R* agrees on the best performing profiles with *Precision@K* and *Recall@K* in almost all cases. Given this, I found it sufficient to only report *Precision@R*, in addition to *Enrichment* at the 99.5% percentile. *Hit@K* tends to disagree with the majority of the other metrics more than the other metrics do and it often also fails to differentiate between feature extraction strategies as well as the other metrics do. Nonetheless, I compute *Hit@K* for all experiment and report it at specific instances.

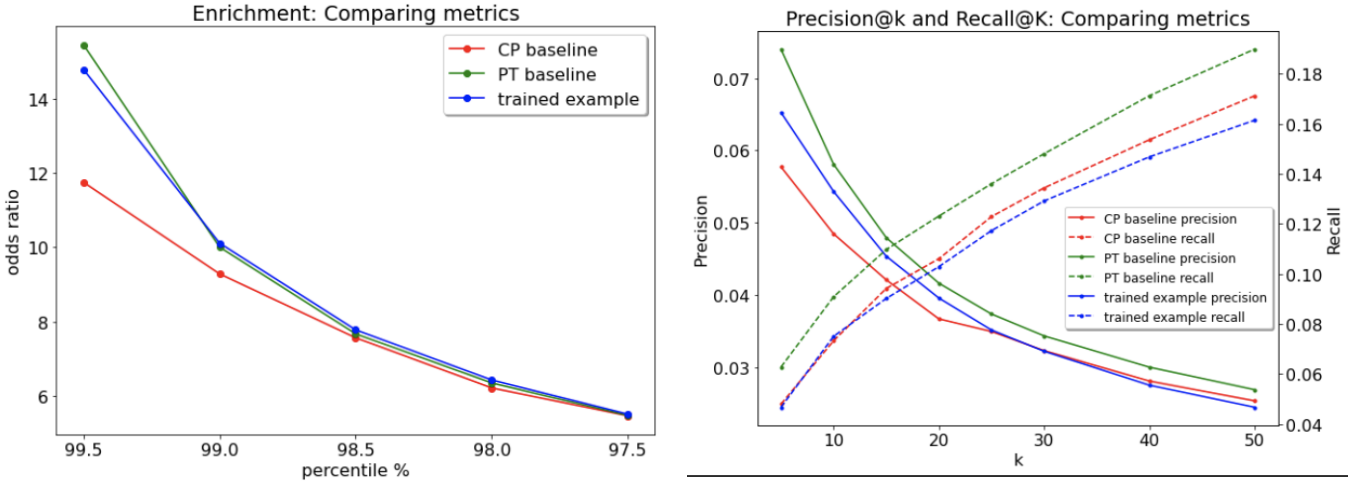

Figure 4.1.: *Comparing the performance plots of three different profiles.* Left: *Enrichment* scores plot the odds ratio at different threshold values between the 97.5 and 99.5th percentile. Right: Precision@K and Recall@K are combined in one plot. percentage values are indicated as fractions, i.e., 0.05 on the precision axis refers to 5%. Experiment IDs: example trained ID: 813, PT baseline ID: 200, CP baseline ID: 100.

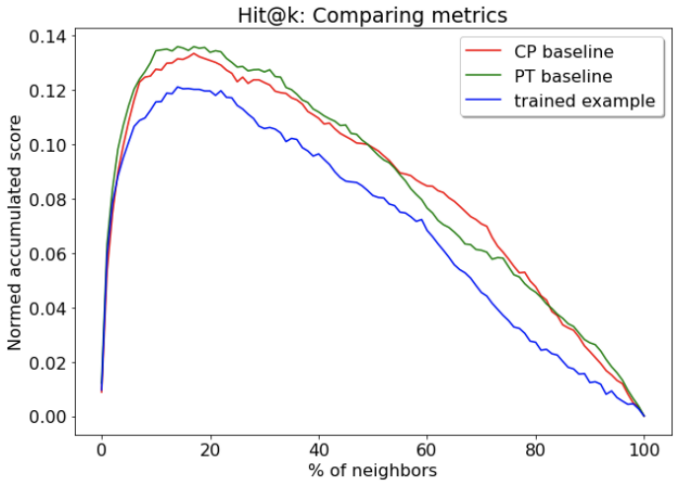

Figure 4.2.: *Comparing the performance plots of three different profiles.* Hit@K figures plot the normalized accumulated scores as described in the *Methods and Materials* chapter 3. The x-axis describes the distance at which the hit was scored: 20% refers to a hit occurring at a fifth of the full list length of neighbors. Experiment IDs: example trained ID: 813, PT baseline ID: 200, CP baseline ID: 100.

Name	Odds ratio at 99.5 %	Precision@R
CP baseline	11.75	0.067
Pre-trained (PT) baseline	15.45	0.072
Example trained	14.74	0.061

Table 4.1.: *Comparing the odds ratio at 99.5% and thePrecision@Rscore of three different profiles.* This table complements the above figures 4.1 showing the exact *Enrichment* values at the highest threshold and the *Precision@R* values. Experiment IDs: example trained ID: 813, PT baseline ID: 200, CP baseline ID: 100.

Link to Discussion 5.2

4.3. Downstream Pipeline

In the *Methods and Materials* 3 chapter, I described the downstream functions available through Pycytominer. The following section describes my research to find the pipeline with the best performance. I compare the pipelines for CellProfiler, pre-trained, and a trained output (ID: 1008) and choose the best one to use in all the following sections.

4.3.1. CellProfiler

Non-spherized data

The LINCS repository has non-spherized Level 5 signatures readily available as the baseline data for CellProfiler data. The downstream pipeline leading to Level 5 profiles can be followed in Figure 1.1 (*Introduction* chapter). Briefly, the steps are aggregation, *Mad robustize* normalization, feature selection, and median aggregation to Level 5 consensus data. The *Mad robustize* and feature selection operations

improve the profiles' performance considerably. Nonetheless, I was able to further improve the Level 5 data by adding an *outlier feature removal* operation. This function removed features with extremely high values that sometimes appear during the normalization process. I determined the best cutoff value for the *outlier feature removal* (OFR) operation to be at 40, thus acquiring the best non-spherized profiles - the *Mad robustize baseline* (ID: 101).

Spherized data

The spherized profiles pipeline differs from the above described *Mad robustize* pipeline by adding the spherizing operation after *Mad robustizing* and feature selection. In this case, I chose not to apply the outlier feature removal operation because it becomes much less relevant for spherized data (see table below 4.2).

Name	odds ratio at 99.5 %	Precision@R
Spherize without outlier feature removal (OFR)	11.75	0.056
Spherize with OFR	11.95	0.056
Mad robustize baseline with OFR	13.48	0.067

Table 4.2.: *Performance comparison of the best CellProfiler profiles. Enrichment* and *Precision@R* values are the highest for Mad robustize data with outlier feature removal at the consensus level (Level 5). Spherized data only marginally improved from OFR. Experiment IDs: Spherize CP profiles, ID 100; Spherize CP profiles with OFR, ID 100; Mad robustize CP with OFR, ID 101.

Set spherized data as baseline

Even though the non-spherized *Mad robustize* data slightly outperforms the spherized data, I decide to use the spherized data as the baseline for CP data for the following reasons. First, all DeepProfiler data performs considerably better with spherized data and this allows me to be more consistent with describing the pipeline and data. Secondly, the improvement in the evaluation metrics is not too large and definitely not larger than deep learning based strategies. Finally, the goal is to make the pipeline more robust and less specific. It is important to point out that *Mad robustize* data performs far worse if the final *outlier feature removal* is not employed. From now on, the CellProfiler baseline refers to the spherized profiles without OFR (ID: 100).

4.3.2. Pre-trained profiles

Unlike with the above CellProfiler profiles, there is no prior experience or set of standards on how to best process the output from pre-trained neural nets. For that reason, I test several pipeline options and report the 99.5% odds ratio and the average Precision@R. Beginning with only the aggregation from well to compound level, the first row of Table 4.3 shows the scores for the most straightforward pipeline. The highest score can be achieved by running the same pipeline as for CellProfiler: *Mad robustize*, feature selection, and spherizing the profiles.

Following a similar argument as in the section above for CellProfiler data, I again decide to take the simplest high-performing pipeline and only use spherizing on pre-trained data. Even though applying Mad Robustize and feature selection before spherizing slightly improve the performance, there is no mathematical motivation for these steps and again, the goal is to find broadly applicable and more robust pipelines.

Name	Odds ratio at 99.5 %	Precision@R
None	1.94	0.0412
Robustize over all	5.14	0.0496
Robustize over DMSO	5.77	0.0498
Mad robustize over DMSO	5.54	0.0497
Mad robustize, Feature select	5.91	0.0473
Standardize over DMSO	6.32	0.0495
Mad robustize, Spherize	15.45	0.0722
Standardize, Spherize	15.45	0.0722
Spherize	15.45	0.0722
Mad robustize, feature select and spherize	16.18	0.0736

Table 4.3.: *Pipeline comparison for profiles from the pre-trained strategy. Enrichment* and *Precision@R* values agree nicely on the order of best performance (increasing performance from top to bottom). Note that the order in which an operation is mentioned in the left column is the order in which they are executed, e.g., "Mad robustize, Feature select " means that first *Mad robustize* normalisation and then feature selection is applied. Spherizing has the largest influence on performance. All pipelines are based on the pre-trained baseline profiles from EfficienNet. Experiment IDs: PT baseline, ID 200.

4.3.3. Trained profiles

The performance of profiles from the training strategy strongly depends on training parameters and training. I have thus tried to choose "representative" training profiles (ID: 1008) to be representative. The pipeline analysis process from the pre-trained profiles is performed here as well (Table 4.4):

Name	Odds ratio at 99.5 %	Precision@R
No normalization	3.73	0.0416
Mad and Feat select	3.99	0.0413
Robustize over all	5.14	0.0440
Robustize over DMSO	5.64	0.0444
Mad Robustize	5.36	0.0441
Standardize	7.11	0.0464
Mad, feature selected and spherize	12.1	0.0413
Mad and spherize	13.37	0.0525
Standardize and spherize	13.37	0.0525
Spherize	13.37	0.0525

Table 4.4.: *Pipeline comparison for profiles from the trained strategy* - equivalent to . Again, *Enrichment* and*Precision@R*values agree on the order of best performance (increasing from top to bottom). Note that the order in which an operation is mentioned in the left column is the order in which they are executed. All pipelines are based on "representative" trained profiles. Experiment IDs: representative trained profiles, ID 1008.

The fact that the feature selected operation does not improve the performance can be observed in all trained profiles. Apart from that, the pipeline results for trained profiles look very similar to those of the pre-trained profiles.

Summarizing the decisions made: I will employ the normalized, feature-selected and spherized consensus data from the LINCS repo as the CP baseline and simply use spherizing for the pre-trained and trained profiling strategy.
Link to Discussion 5.3

4.4. Feature space

In the following sections, I present PCA plots of the profile feature space for all three profiling strategies and investigate the explained variance of the first two principal components (PCs) and how negative control wells are distributed among the perturbed wells. The following plots all show the first two PCs of Level 3 profiles (non normalized well-level profiles), i.e., every point is a well (blue: DMSO and orange: perturbed). Recognizing the limitations of a 2D visualization of high-dimensional data, I use these plots to identify the distribution between negative controls and perturbations.

CellProfiler

Figure 4.3 shows the feature space of the non-normalized and non-spherized Level 3 CellProfiler data via a PCA plot. The first two principal components explain 83%, and 8% of the total variation in the data. These high values are likely due to strong technical artifacts and highly-correlated features present in the raw data. After spherizing (where the transformation is learned on the negative controls alone), the explained variation drops to 28% and 5% (PCA plot not shown) - a sharp decrease and reminder of how spherizing reduces correlations in the whole dataset, not just the data on which it was trained. The median of the median of all CP features is 0.9 but the mean of the mean of all features is 64. This shows how a few features have very high positive values.

Pre-trained

After aggregating single-cell profiles from the pre-trained EfficientNet and ResNet models to the well level, we can investigate the feature space via a PCA plot. The first two principal components explain 53% and 15% (EfficientNet) and 44% and 18% (ResNet) of the variation in the dataset. Interestingly, the scatter plots align visually although the magnitudes are different and the y axis is flipped for EfficientNet. The highest value found among all features and wells was 4.8 and the lowest value found was -0.25. Note that these profiles are considerably more centered around zero than the CP profiles.

Trained

The trained profiles' feature space looks a lot more circular and centered than the other two spaces. However, the DMSO wells are still distributed throughout the perturbed space and do not center around a point like one would expect without technical artifacts. The explained variation lies at 4.9% and 4.0%, considerably lower than all other profiling strategies. The means of all features follow a normal distribution around 0.2.

Link to Discussion 5.4

4.5. Baseline performance – Pre-trained nets and CellProfiler

When performing spherizing on the ResNet profiles, I encountered a numerical barrier. This is due to the fact that some features have such a low variance that dividing by them leads to numerical instabilities. This is solved by applying a feature selection process prior to the spherizing operation. Figure 4.5 and

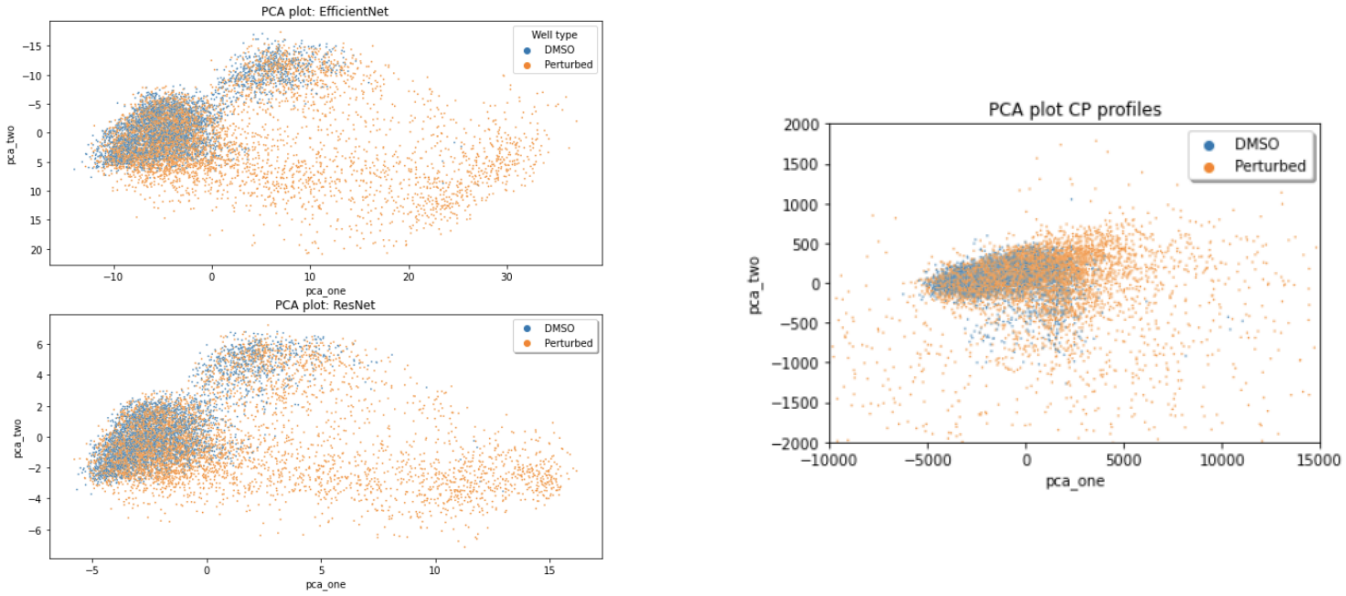

Figure 4.3.: *PCA plot of the first two principal components.* Left: Each point represents a well profile. Note that the range of scattered points goes well beyond the limits of this figure. Right: PCA plots of EfficientNet and ResNet pre-trained profiles. Note that the y-axis of the EfficientNet plot has been reversed. Experiment IDs: CP baseline ID 100; EfficientNet, ID 200; ResNet, ID 201.

Table 4.5 display the performance comparison between the CellProfiler profiling strategy (CP baseline) and the profiles of the two pre-trained architectures. Both deep learning model strategies clearly outperform in the *Enrichment*, *Precision@K* and *Recall@K* metrics as well as for *Precision@R*. *Hit@K* does not show any difference among the three models.

Name	Odds ratio at 99.5 %	Precision@R
EfficientNet	15.45	0.072
ResNet	15.45	0.069
CP baseline	11.75	0.056

Table 4.5.: *Comparing performance of EfficientNet and ResNet pre-trained profiles to the CP baseline.* Supplement table to Figure 4.5. Experiment IDs: CP baseline ID 100; ResNet pre-trained, ID 201; EfficientNet pre-trained, ID 200.

Choosing EfficientNet

The two architectures perform very similarly: while ResNet has better *Enrichment* and *Precision@R* scores, *Recall@K* scores for EfficientNet are stronger. I therefore chose to compare all trained model performances to the pre-trained EfficientNet (PT baseline | ID: 200). This is due to the fact that most training experiment are run on EfficientNet and that the ResNet profiles require an additional operation and potentially would perform more poorly than than EfficientNet without the feature selection step.

Link to Discussion 5.5

4.6. Baseline performance – Subsets

Motivation for subsets

Training on the full set of 10 million cells takes a considerable amount of time, even with the fastest

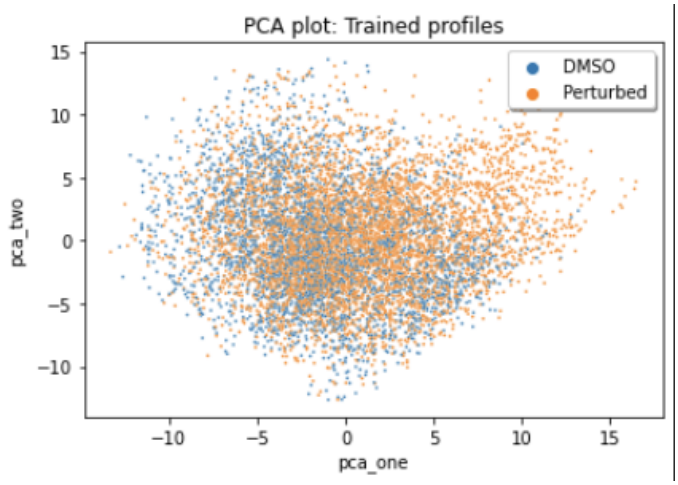

Figure 4.4.: PCA plots of the first two principal components of an example trained profile. Experiment IDs: example trained profiles, ID 813.

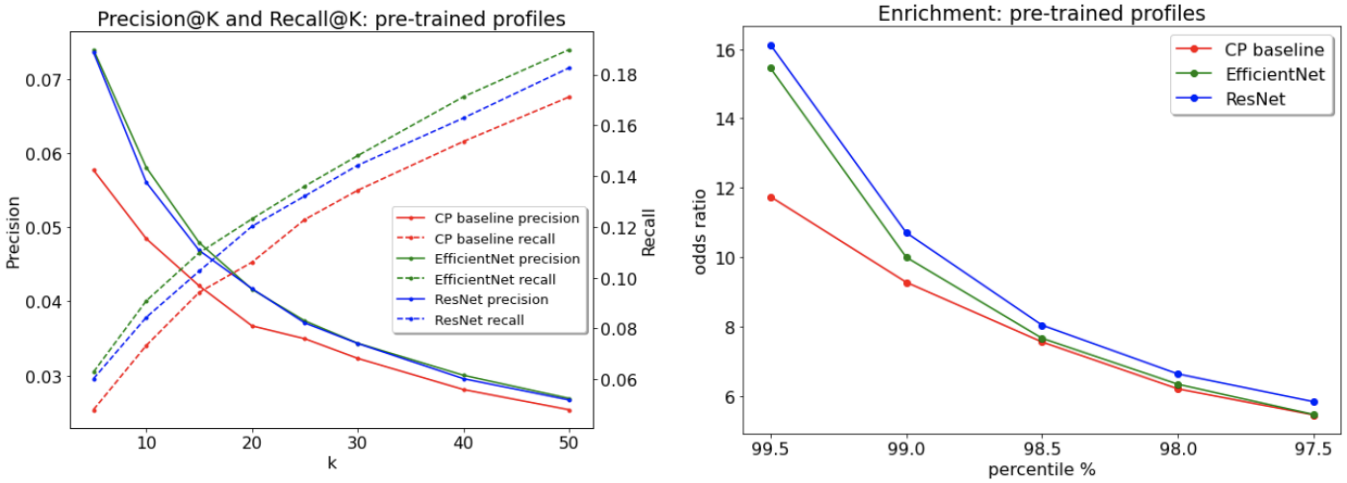

Figure 4.5.: *Comparing performance of EfficientNet and ResNet pre-trained profiles to the CP baseline.* Both pre-trained models clearly outperform the CellProfiler baseline. Experiment IDs: CP baseline ID 100; ResNet pre-trained, ID 201; EfficientNet pre-trained, ID 200.

GPUs. To mitigate this problem, I chose to either train on a few cells per well or to train on only a subset of all compounds and wells. Secondly, I am interested in gaining a more precise understanding of the performance of different compounds, as the evaluation metrics give no indication of whether certain compounds score higher or lower than the average. In the following section, I describe the subsets chosen and then investigate their performance.

4.6.1. Data subsets

Top 50 MAOs

The Top50 subset is motivated by the fact that there is a large variety in MOA classes and I want to investigate how the MOA class size affects the performance. The 50 MOAs with the largest number of compounds linked to them were selected to be in the Top50 subset. This splits the data equally into two subsets of ∼550 compounds, representing 50 MOAs in the in-Top50 set and ∼180 MOAs in the

out-Top50. I will refer to the subset of MOAs and compounds in the Top50 group as in-Top50 subset and the complementary MOAs as out-Top50.

Strong Phenotype
The Strong Phenotype (SP) subsection includes those Compounds that have the strongest phenotypic response, that is, correlate the least with DMSO. To find these compounds that are most different from DMSO, I simply calculated the Pearson correlation coefficient for all samples relative to DMSO on the spherized pre-trained dataset and kept the ∼450 least correlated compounds. Again, the dataset is split into two subsets: in-SP containing ∼450 compounds and out-SP containing ∼700 compounds.

4.6.2. Performance

Calculate metrics on subsets
Since *Enrichment* and *Hit@k* are global metrics, the same function could be applied to the smaller subsets to achieve valid results. For *Precision@k* and *Recall@k*, this is not possible, as the MOA class size must be more than one for the score to be mathematically defined. Thus, I calculate these two scores on the full dataset and then simply average the precision and recall values for each subset. In other words, I compute the compound-level *Precison@K* and *Recall@K* using the whole dataset, and then average these values for each subset.

In the following Figures (4.6 4.7 4.8) , I take the EfficientNet pre-trained profiles and run the evaluation metrics for both Top50 and SP subsections. performing the same experiment on the CP baseline data leads to equivalent results to those shown below.

Expectations for metrics
Mathematically, we would expect *Precision@K* to increase for the Top50 subset and *Recall@K* to remain the same. Assuming random noise profiles, the probability of MOA matches is evenly distributed between the first and the last nearest neighbor. For compounds with several MOA matches, the probability of a match occurring in the first k neighbors increases linearly with the number of available MOA matches. Thus we expect *Precision@K* to be higher for the in-Top50 subset. On the other hand, we expect Recall@K to be equal for in/out and the full subset because the probability of matches below and and above k is constant regardless of the MOA class size. The *Enrichment* score is also expected to be independent of MOA class sizes. Here, the same argument as for *Recall@K* stands: the probability to find MOA pairs is equal above and below the threshold and thus we expect an odds ratio of one for randomized data. We do not expect any of these effects in the SP subset because the distribution of large MOAs in this subset is similar to the full dataset. I have numerically tested the above predictions to be valid with random sets of data.

Figures 4.6 (left) compare the performance of the in-Top50 subset, the out-Top50 subset and the full dataset. Clearly, the in-Top50 set performs better on all metrics (apart from *Hit@K*) and the full data set outperforms the out-Top50 dataset. In the case of the Strong Phenotype subsections, as expected performance increases dramatically for compounds with strong phenotypes - as can be seen in Figures 4.6 (right).

Conclusion
The overlap between the two subsections was found to be 225 compounds. In other words, 225 of the 520 compounds in the Top50 set also were in the SP set. This is approximately what one would expect from

a random selection of two subsets. Note that this experiment is not only important to learn about the differences in compound performance but also to prepare for training on these subsets of data. I will show that training on a given subset of data will improve the performance of that subset. Hence these pre-trained evaluation metrics serve as a baseline to the upcoming performance scores of the trained profiles (see section *Training on subsets* 4.7.6).

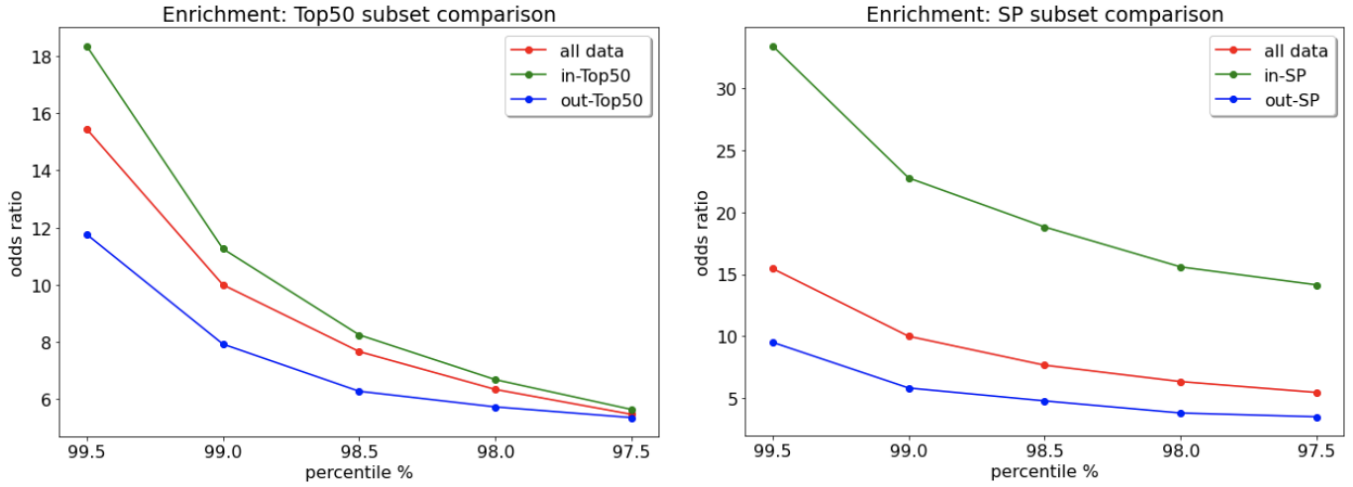

Figure 4.6.: Left: *Comparing performance of the in-Top50, out-Top50 and full data sets.* Note, *Enrichment* and *Hit@K* are calculated over the subset while *Precision@K* and *Recall@K* are computed on the full dataset, then averaging the values for each subset. Further note, that all red lines in the images on the left and right are the same. Right: *Comparing performance of the in-SP, out-SP and full data sets.* In-SP subset performs better than any other profiles seen this far. All data is based on the PT baseline. Experiment IDs: PT baseline, ID 200.

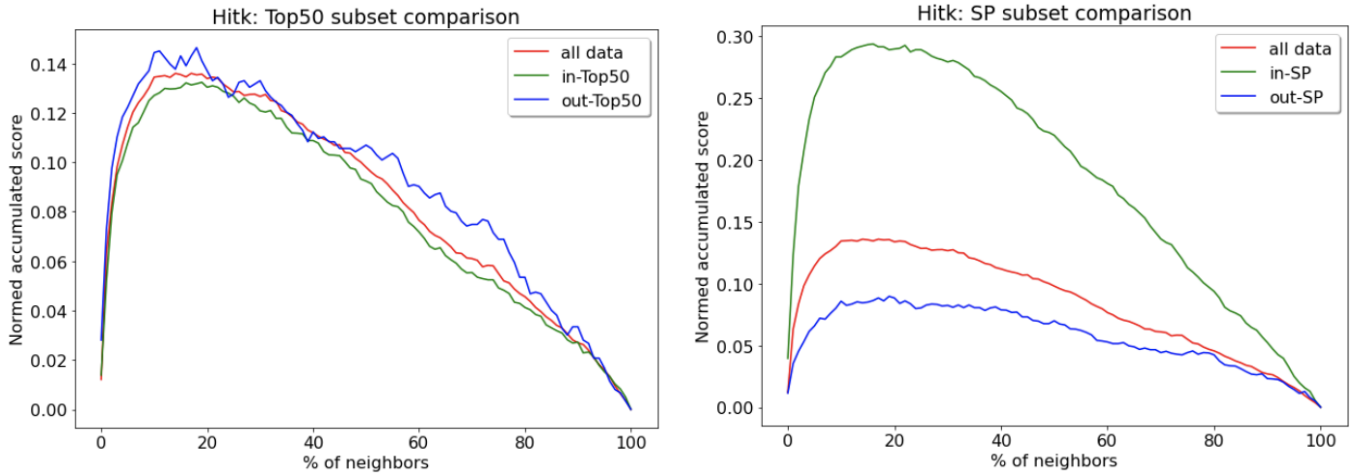

Figure 4.7.: See above

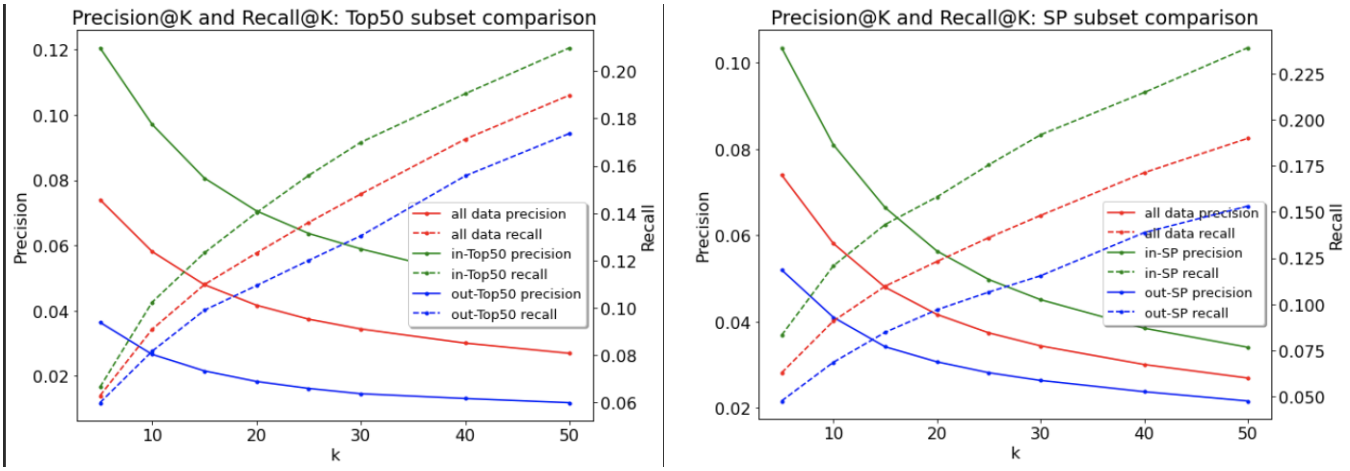

Figure 4.8.: See above

Link to Discussion 5.6

4.7. Training models

Finally, we turn our attention to training models as a profile extraction strategy. In this section, I will report how different training parameters influence the training accuracy and performance of a model. When referring to the performance of a model, I simply mean the performance of the resulting feature space on the primary task of grouping compounds of the same MOA together. Note that while training loss, accuracy, and validation accuracy report on the auxiliary compound classification task, the MOA-centric evaluation metrics (discussed in the *Metric comparison* section) report on the primary task (assigning an MOA class to compounds). Increasing the auxiliary performance (training and validation accuracy) is often beneficial to maximizing the primary evaluation metrics.

Note: The training and validation accuracy and performance highly depend on the training sample, i.e., the number and type of cells that the model trains on. Since I used different training datasets, some of the accuracy and performance values reported in this chapter will differ widely. This point will be addressed in the *training on subsets* section 4.7.6. The exact training parameters for each of the presented experiment can be found via the Experiment ID.

The relevant parameters and their typical value range for training are:

- Architecture: [EfficientNet, ResNet]
- Training data: Number of cells, number of compounds, batches, etc.
- Initialization: [ImageNet, Random]
- Augmentation: [True, False]
- Label smoothing: [0-0.2]

- Epochs: [10-30]
- Batch size: [32 - 256]
- Learning rate: [0.0001 - 0.05]
- Learning rate schedule: [flat, cosine]

4.7.1. Learning rates and schedules

Figure 4.9 presents the development of the training accuracy and validation accuracy during the training experiment. Training runs are marked with different colors while the training accuracy is represented by a full line and the validation accuracy by a dashed line.

Issues of constant learning rates

Figure 4.9 shows a comparison of a constant (flat) learning rate (LR) of 0.005 and a cosine learning rate schedule initialized at 0.005. As a brief reminder: the LR rate will grow linearly in the first five epochs and then decay like a cosine function while the flat LR is always the same. While the flat learning rate converges much more quickly at the beginning, the end result is very similar. Additionally, other experiment show that a high learning rate can lead to unstable validation accuracy values. They are considered unstable because they do not properly converge to a value at high epoch numbers (see *Appendix* 6). In other experiments (not shown here) I found that a custom learning rate schedule can match the performance of the cosine learning rate; however, this is unnecessary complexity. I have thus employed the cosine learning schedule for most experiment to ensure stable convergence at higher epochs. The performance between a flat and cosine schedule that arrive at similar accuracies is slightly in favour of the cosine schedule (see Table 4.6).

Name	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
Flat learning rate	8.68	0.046	0.981	0.152
Cosine learning rate	10.62	0.045	0.967	0.170

Table 4.6.: Left: *Comparing the performance of learning rate schedules.* Both experiment had a learning rate value of 0.005. Complimentary table to Figure 4.11 (left). Experiment IDs: flat learning schedule ID 913; cosine learning schedule, ID 910.

Choosing cosine 0.02

First, I investigated the effect of the learning rate value in a cosine learning rate schedule on the training process in Figure 4.9. A higher learning rate leads to faster learning, i.e., faster convergence for the loss and validation accuracy (VA) and slightly higher performance values (see Table **??**). I found that learning rates above 0.02 do not result in any measurable increase in performance. Instead, higher learning rates lead to unstable convergence. Hence, most experiment were run on a cosine learning rate of 0.02.

Name	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
LR = 0.002	8.88	0.043	0.783	0.297
LR = 0.005	9.71	0.045	0.922	0.313
LR = 0.02	12.58	0.053	0.995	0.331

Table 4.7.: *Comparing learning rates.* Higher learning rates increase performance. Complimentary to Figure 4.11 (right). Experiment IDs: LR = 0.02, ID 927; LR = 0.005, ID 917; LR = 0.002, ID 929

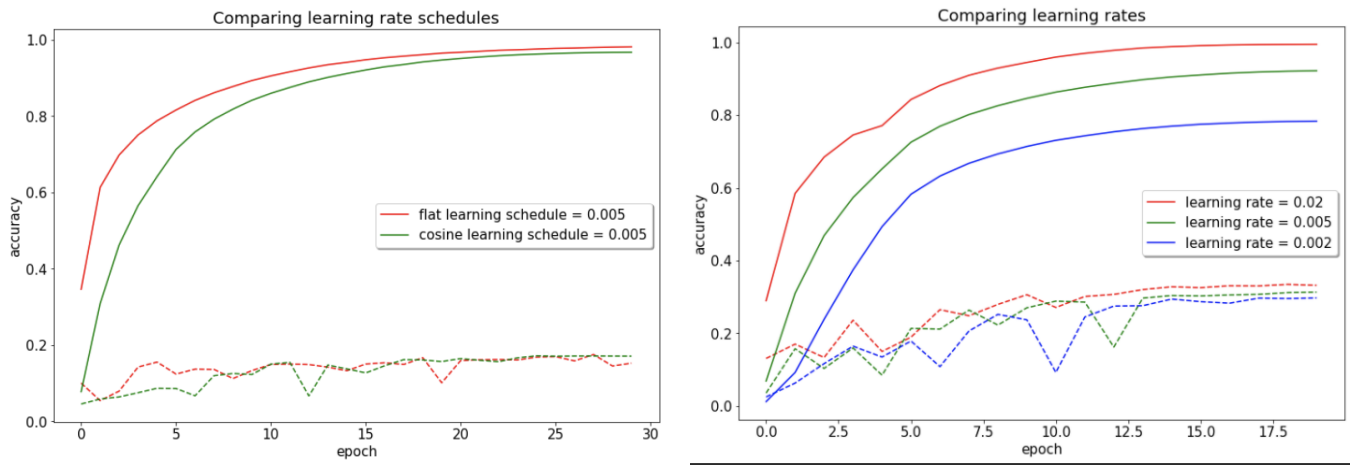

Figure 4.9.: Left: *Comparing learning rate schedules.* The training accuracy (full line) and validation accuracy (dotted line) for both the constant and cosine learning rate of 0.005 are very similar. Experiment IDs: flat learning schedule ID 913; cosine learning schedule, ID 910. Right: *Comparing learning rates.* Higher learning rates increase the training (full line) validation accuracy (dotted line). Experiment IDs: LR = 0.02, ID 927; LR = 0.005, ID 917; LR = 0.002, ID 929.

Link to Discussion 5.7.1

4.7.2. Batch sizes

In this subsection, I investigate the effect that the batch size has on training and performance. Each epoch of training is split into s number of batches each holding b_c cell samples such that $stimesb_cN$ where N is the total number of cells in the training set. Thus smaller batch sizes increase s the number of gradient descents performed within one epoch. Figure 4.11 and Table 4.9 show the comparison of two training runs that are identical apart from the batch size. This clearly shows how a large batch size such as 256 does not allow the model to converge as well as the smaller batch sizes. We can observe a clear performance increase in Figure 4.10 for the models trained with smaller batches. However, the smallest batch size (32) does not further increase the validation accuracy.

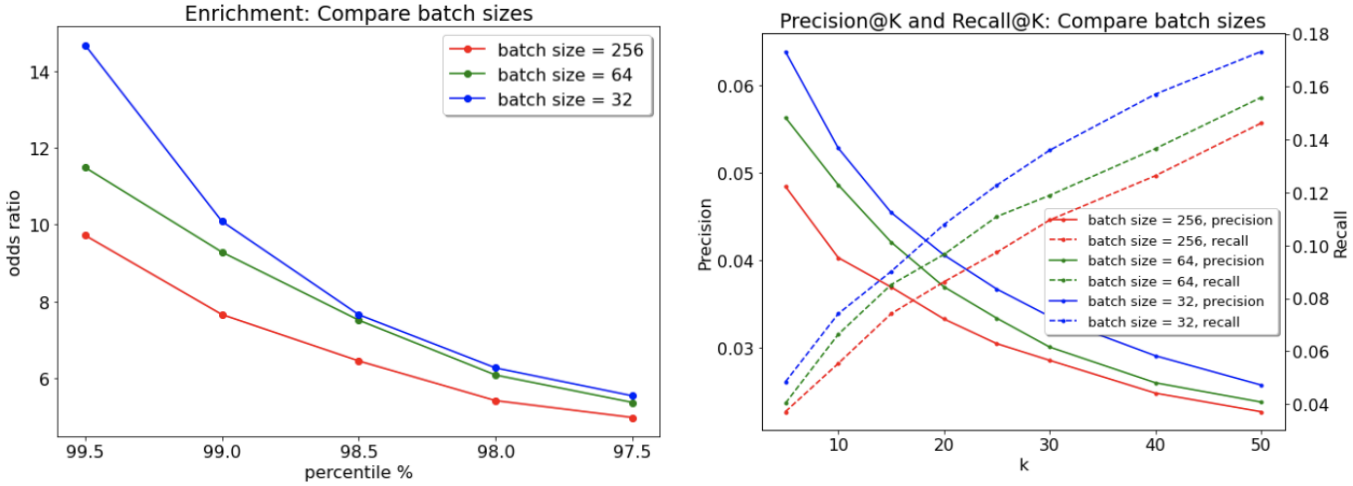

Figure 4.10.: *Comparing batch sizes.* Experiment IDs: BS = 256, ID 917; BS = 64, ID 918; BS = 32, ID 925.

Name	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
Batch size = 256	9.71	0.045	0.922	0.313
Batch size = 64	11.48	0.052	0.985	0.342
Batch size = 32	14.67	0.060	0.986	0.296

Table 4.8.: *Comparing batch sizes.* Complimentary table to Figure 4.10. Experiment IDs: BS = 256, ID 917; BS = 64, ID 918; BS = 32, ID 925.

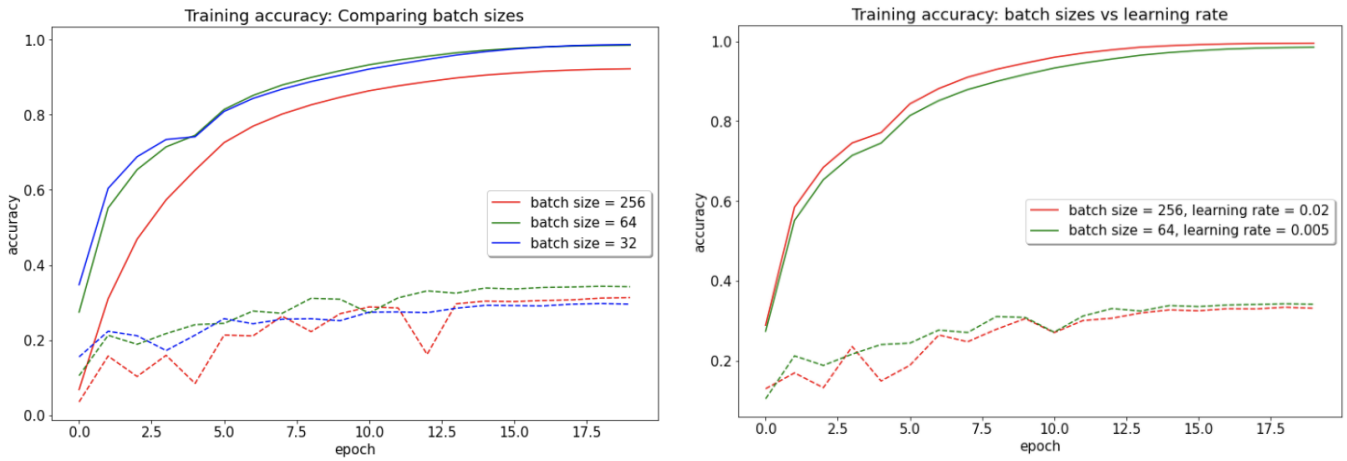

Figure 4.11.: Left: *Comparing batch sizes.* Experiment IDs: BS = 256, ID 917; BS = 64, ID 918; BS = 32, ID 925. Right: Comparing batch sizes and learning rates. The two parameters counteract each other, leading to very similar accuracies and performances. Experiment IDs: BS = 256, ID 927; BS = 64, ID 918.

BS and LR interaction

Combining the learnings from the learning rate and batch size section, I asked myself if and how these two parameters counteract one another. Increasing the learning rate and increasing the batch size should have a similar effect to decreasing the learning rate and decreasing the batch size. In this final Figure 4.11 (right), I show how the combination of BS = 256 and LR = 0.02 resembles the combination of BS = 64 and LR = 0.005. However, the former combination shows somewhat higher evaluation scores.

Name	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
Batch size = 256 *Learning rate = 0.02*	12.58	0.053	0.9952	0.331
Batch size = 64 *Learning rate = 0.005*	11.48	0.052	0.985	0.342

Table 4.9.: *Comparing batch sizes and learning rates.* The two parameters counteract each other, leading to very similar accuracies and performances. Experiment IDs: BS = 256, ID 927; BS = 64, ID 918

Link to Discussion 5.7.2

4.7.3. Epochs

Generally speaking, training on more epochs improves the model's performance up to a converging value. Table 4.10 shows the performance of the profiles that all originate from one training run. From this 25 epoch training, the intermediate models after 8, 16 and 25 epochs were used for profiling. Training on more epochs increases the *Enrichment* score but less so Precision and Recall. Epochs' increasing performance is a trend visible among all experiment, even when other parameters are tweaked.

Convergence
Furthermore, I have found evidence of increasing performance beyond the visible point of convergence, albeit small. In a large training set (Table 4.11), the model seemed to have converged at epoch 12 with a training accuracy of 0.989 and a validation accuracy of 0.275. In the following eight epochs, the model only increased training accuracy to 0.995 while validation accuracy did not increase (0.269).

Name	Odds ratio at 99.5%	Precision@R
8th epoch	10.21	0.0473
16th epoch	12.42	0.0474
25th epoch	13.26	0.0490

Table 4.10.: *Performance after the Nth epoch.* Experiment ID: 827/8,16,25.

Name	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
12th epoch	11.17	0.0573	0.989	0.275
20th epoch	13.48	0.0590	0.995	0.269

Table 4.11.: *Performance after the Nth epoch.* Experiment IDs: 12 epochs, ID 930; 20 epochs, ID 931.

Link to Discussion 5.7.3

4.7.4. Label smoothing

Increasing the label smoothing (LS) variable has a direct effect on loss function and thus the way the model converges to a minimum. Figure 4.12 shows four different trainings with label smoothing variables from 0.0 to 0.2. The higher the LS, the higher the loss value; the training accuracy, however, does not correlate in order with the LS variable. The validation accuracy remains unaffected by the changes of the LS. Surprisingly, the *Enrichment* score of the $LS = 0.1$ shows a noticeable increase over the other three models but this higher performance is not apparent in the precision and recall plots (Table 4.12).

Name	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
Label smoothing = 0.0	9.76	0.0525	0.936	0.665
Label smoothing = 0.05	9.42	0.0493	0.919	0.649
Label smoothing = 0.1	11.90	0.0550	0.755	0.640
Label smoothing = 0.2	10.01	0.0510	0.871	0.646

Table 4.12.: *Comparing label smoothing parameters.* Complimentary table to Figure 4.12. Experiment IDs: LS = 0.0, ID 1008; LS = 0.05, ID 1016; LS = 0.1, ID 1012; LS = 0.2, ID 1009.

Link to Discussion 5.7.4

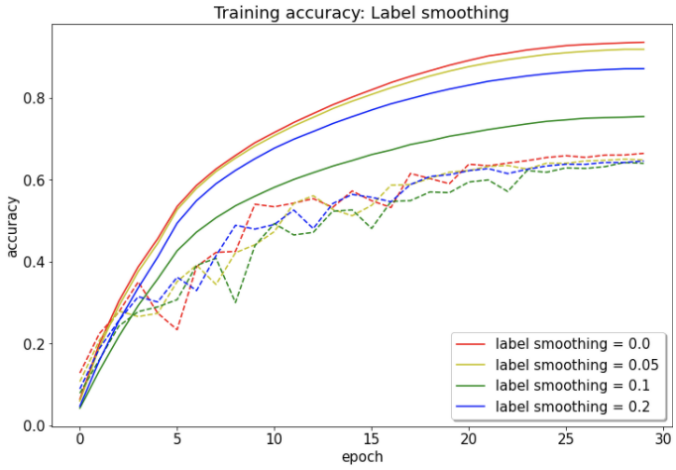

Figure 4.12.: Comparing label smoothing parameters. Experiment IDs: $LS = 0.0$, ID 1008; LS $= 0.05$, ID 1016; $LS = 0.1$, ID 1012; $LS = 0.1$, ID 1009.

4.7.5. Augmentation

Older versions of DeepProfiler did not include the option to augment images at runtime. Due to the need for better regularization tools, we implemented the augmentation option. Enabling augmentation decreases the training and validation accuracy gap and is thus successful at regularizing the model (see Figure 4.13). In this experimental comparison, using augmentation increases performance as can be seen in Table 4.13. Unfortunately, augmentation does not always increase performance for all experiments, as can be seen in the following subsection.

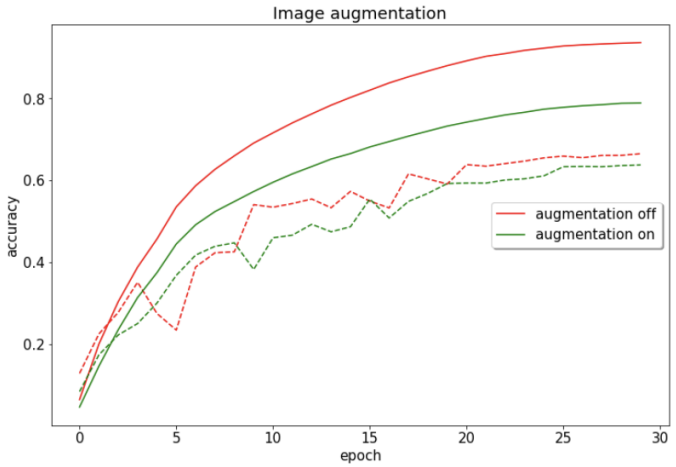

Figure 4.13.: *The effect of augmentation on training accuracy.* Experiment IDs: augmentation off, ID 1008; augmentation on, ID 1010

Name	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
Augmentation off	9.76	0.0525	0.936	0.665
Augmentation on	12.89	0.0557	0.788	0.637

Table 4.13.: *The effect of augmentation on performance.* Experiment IDs: augmentation off, ID 1008; augmentation on, ID 1010.

Augmentation and label smoothing

The previous two sections concluded that a label smoothing value $= 0.1$ and image augmentation increases the performance of models. Subsequent experiment show, however, that the interaction between these parameters is less straightforward. In one experiment, the combination of augmentation on and $LS = 0.1$ results in a worse performance than the same experimental setup but with augmentation off / $LS = 0.1$. Knowing that this discrepancy was due to the models not fully converging, I ran further experiments with higher learning rates and lower batch sizes to ensure full training convergence. Table 4.14 shows a comparison of augmentation and label smoothing switched on and off. The influence of label smoothing and augmentation seems to be minimal for these high performing training experiment.

In conclusion, the data seems to show that for very high-performing models that are close to convergence, augmentation and label smoothing has less of an effect than for low-performing models. Unfortunately, this does not allow me to conclude which parameters to use to achieve the highest performance.

Name	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
Augmentation off *Label smoothing = 0.0*	16.349553	0.0694	0.9365	0.6511
Augmentation on *Label smoothing = 0.0*	15.339429	0.0582	0.8947	0.6316
Augmentation on *Label smoothing = 0.1*	16.179841	0.0701	0.8768	0.6372

Table 4.14.: *The effects of augmentation and label smoothing on high performing models.* Experiment IDs: augmentation off & LS = 0.0, ID 1103; augmentation on & LS = 0.0, ID 1105; augmentation on & LS = 0.1, ID 1028.

Link to Discussion 5.7.5

4.7.6. Training on subsets

In the section *Baseline performance - Subsets*, I showed that the Top50 and Strong Phenotype (SP) subsets perform above average. I now investigate the performance of models that have been trained on Top50 and SP subsets to the known performances of the pre-trained model and models trained on the full dataset.

Top50

First, I present the performance of in- and out-Top50 subsets using a model (trained on Top50) after eight epochs and after 25 epochs (see similarity to the *epochs* section 4.7.3). In Figure 4.14, we can observe an increased *Enrichment* on the compounds that have been trained on, while the untrained compounds do not show any improvement. However, the *Precision@K* and *Recall@K* scores do not increase from the training - quite similar to the effect seen in Table 4.10 (in the *epochs* section 4.7.3). We learn: through training, the model learns the representations of the in-Top50 compounds such that the *Enrichment* score increases.

SP shows strong increase
As noted in the section *Baseline performance - Subsets* (Figure 4.6), the Strong Phenotype subset exhibited

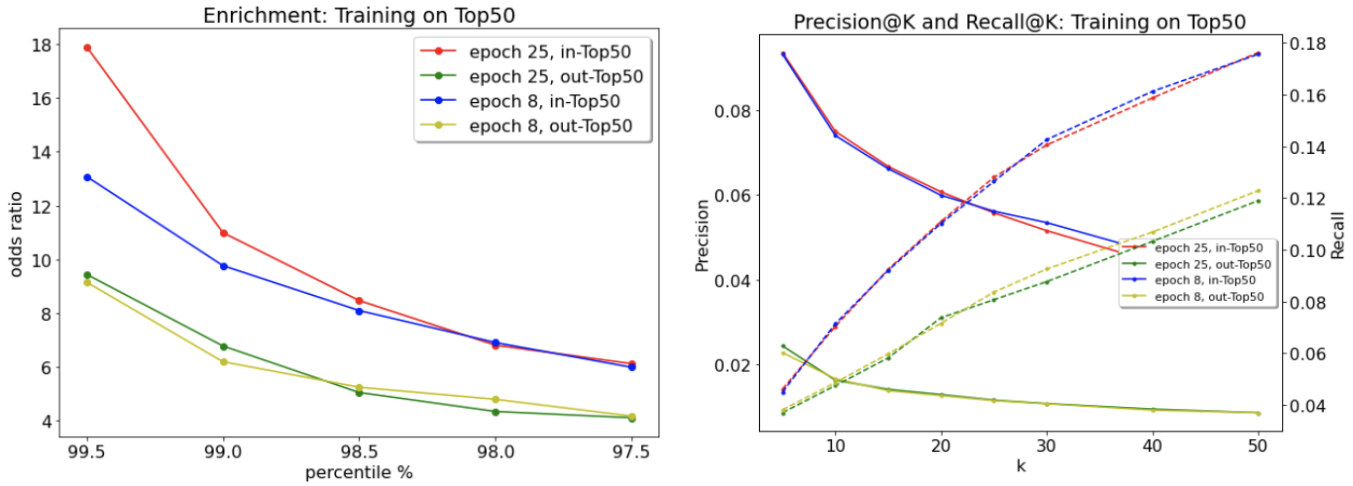

Figure 4.14.: *Training and evaluating the in-Top50 set.* Two profiles (from the 8^{th} epoch and from the 25^{th} epoch) are evaluated on the two Top50 subsets. The relevant increase through training can be found between the blue and red line and the green and yellow line. Note that I have omitted the Recall@K (the dashed lines) labels for clarity. Experiment IDs: epoch 8, ID 827/8; epoch 25, ID 827/25.

a larger gap between in-SP and out-SP performance than the in-Top50 subset. Figure 4.15 shows the performance of in-SP and out-SP profiles after being trained on the SP dataset. For reference, I have added in- and out-SP performance of the pre-trained model. The *Enrichment* scores for the trained on SP, in-SP case are larger than all other *Enrichment* scores seen in my project. On the other hand, the out-SP scores decrease through the training down to only 3.99 at the 99.5% percentile. In Figure 4.15, *Recall@K* also shows a strong increase for the in-SP and a decrease for the out-SP profiles. The *Precision@K* scores show only the smallest increase of the in-SP subset and the smallest decrease of the out-SP data, i.e., the *Precision@K* performance is almost unaffected by the training of the SP set. In conclusion, the training of a model on in-SP data strongly improves the *Enrichment* and *Recall@K* scores while decreasing the performance of the untrained out-SP data. Confusingly, the effect of this training process on *Precision@K* is barely measurable.

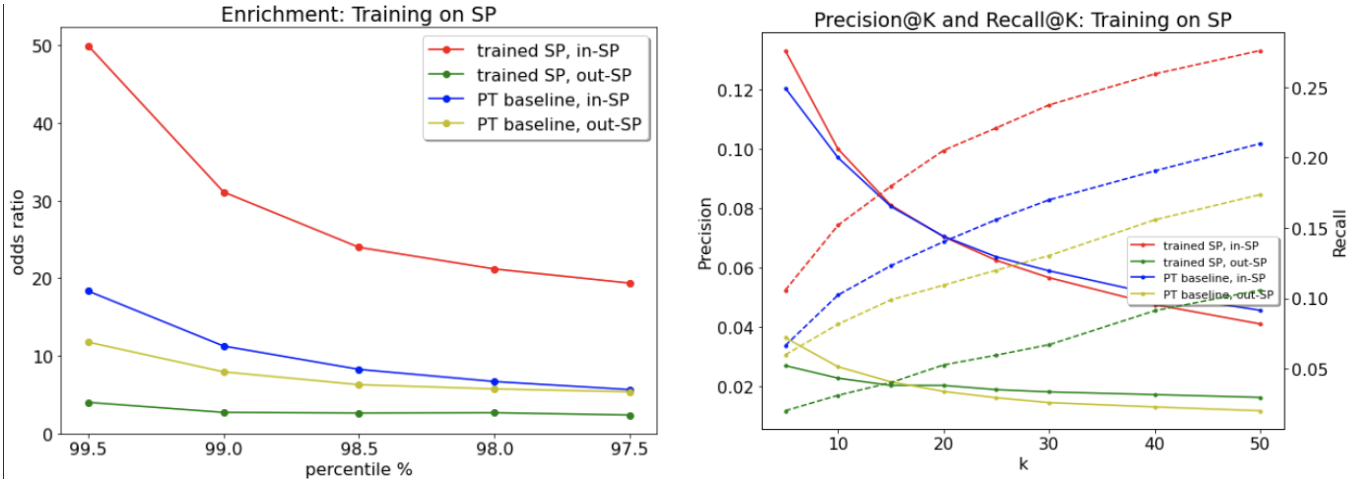

Figure 4.15.: *Training and evaluating on the SP set.* A model trained on in-SP data and the pre-trained baseline are evaluated on the in- and out-SP subsets. The relevant increase through training can be found between the blue and red line and the green and yellow line. Note that I have omitted the Recall@K (the dashed lines) labels for clarity. Experiment IDs: trained SP, ID 926; PT baseline, ID 200.

Training on SP vs all

Next, I compare the SP set performance after the model is trained on all compounds, not only the in-SP set. Figure 4.16 shows how the *Enrichment* score of the SP set also increases in comparison to the pre-trained scores. Even though this model is trained on all compounds, the in-SP set increased in performance. Additionally, I have added the out-SP *Enrichment* for the SP-trained model to this Figure. We can see that training on all compounds does not improve the out-SP compounds, even when compared to a model that was not trained on these compounds at all. Below, I compare the performance of the out-SP dataset between and trained on in-SP and trained on all compounds (Table 4.15.

Name	Odds ratio at 99.5%	Precision@R
out-SP performance, Trained on in-SP	3.996	0.0233
out-SP performance, Trained on all data	4.644	0.0269

Table 4.15.: Out-SP performance. Experiment IDs: Trained on in-SP, ID 926; Trains on all data, ID 921.

Moreover, I further compared the full dataset scores for the SP-trained and all-trained profiles. When trained on all compounds, the full set performance marginally increases. Further investigating the validation accuracy in Figure 4.16 , we notice a clear discrepancy of the validation accuracy between the SP, Top50 and full set training. While the training accuracy is very similar for all three cases, the validation accuracy is considerably higher for the SP training set.

Name	Odds ratio at 99.5%	Precision@R
Trained on in-SP compounds	12.213	0.0535
Trained on all compounds	13.269	0.0596

Table 4.16.: *Full data performance.* Experiment IDs: Trained on in-SP, ID 926; Trains on all data, ID 921.

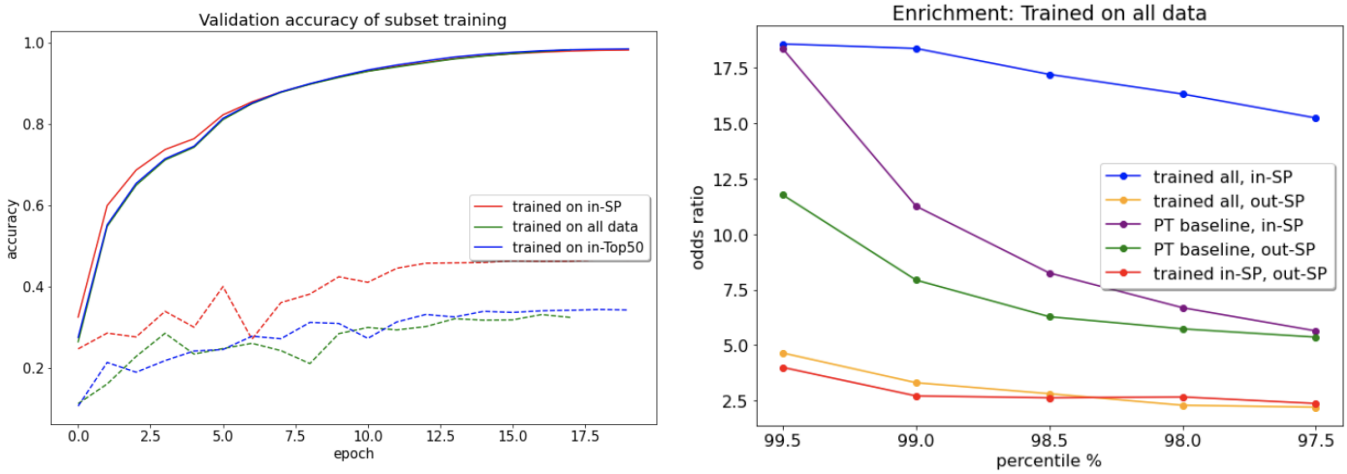

Figure 4.16.: Left: *Validation accuracy for trained SP.* Right: *Enrichment scores for SP sets of trained on all data.*"Trained all, in-SP" refers to the *Enrichment* score of in-SP subset data with a model trained on all compounds (not only the SP data); "PT baseline, in-SP" refers to the scores of in-SP data from the pre-trained baseline model (compare Figure 4.6 from the Baseline performance - Subset section). "Trained in-SP, out-SP" refers to the performance of an in-SP trained model evaluated on the out-SP set. Experiment IDs: Trained on in-SP, ID 926; Trained on all data, ID 921; Trained on Top50, ID 918

Overall, these experiment imply that the ability of models to classify single cells and to perform well on the evaluation metrics highly depends on the phenotype strength of the cell type. Furthermore, training

on the in-SP data increases the performance of that subset of the data by a great deal, however, the total performance decreases when compared to an all data trained model.

Training on all compounds

While training on subsets is faster and yields interesting insights into the effects of training on different types (strong or weak phenotype) of cells, the final goal remains to optimize the performance of the LINCS dataset. Figure compares two models that have been trained with identical parameters with the only difference being the number of compounds in the training set. The first data set contains 500,000 cells from ∼450 randomly selected compounds while the second data set contains the same number of cells from all ∼1,100 compounds. The results confirm that a wider diversity (all compounds) of cells is beneficial to the performance, especially if that wider diversity is found in the evaluation data (see Table 4.17). These results reveal the strategy for obtaining the best models is training and overfitting onto the entire dataset instead of only a subsection of compounds or batches. This strategy is followed in my later experiment, leading to my strongest models (see *High performing models* 4.7.8) and is the approach recommended for subsequent experiment.

Name	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
Data set one 450 compounds	15.117	0.0602	0.9777	0.7237
Data set two 1100 compounds	15.785	0.0615	0.9614	0.6549

Table 4.17.: *Training on more compounds.* Note that the validation data for data set one also only contains cells from the 450 compounds, explaining the higher validation accuracy. Experiment IDs: data set one, ID 1021; data set two, ID 1024.

Link to Discussion 5.7.6

4.7.7. Training on larger datasets

Training on smaller subsets is an important tool to run many experiment and save computational resources. However, Table 4.18 shows that a larger number of cells in the training set increases performance. All three training sets contain 450 compounds with different numbers of cells as shown, taken from the 18 million cells in the entire dataset. The compounds are randomly selected, that is, they have no specific MOA class size or phenotypic relevance. Cells were taken from all 5 technical replicates of each compound and thus every batch was included in this training. The increase in size is achieved by simply dedicating more cells as training cells, instead of unused cells.

Name	Number of cells	Odds ratio at 99.5%	Precision@R	Training accuracy	Validation accuracy
Small	456,533	9.768	0.0525	0.936	0.665
Medium	911,042	10.77	0.0517	0.952	0.782
Large	2,267,640	11.79	0.0559	0.979	0.904

Table 4.18.: *Training on larger datasets.* Experiment IDs: Small, ID 1008; Medium, ID 1019; Large, ID 1020.

Link to Discussion 5.7.7

4.7.8. High performing models

Concluding the *Trained models* section of this chapter, I present the models that achieve the highest performance here. No specific information can be learned from these Figures apart from the fact that I can train models that perform as well as the pre-trained baseline.

Figure 4.17 shows (the well known) performance plots for two trained models that achieve performance scores similar to the pre-trained baseline. Following the trend seen in many other experiments, the trained models slightly outperform the PT baseline in the *Enrichment* score while the opposite is true for *Precision@K* and *Recall@K*. The two trained models have different hyperparameters (for example: model 1 has augmentation and model 2 does not) but they trained on all batches and compounds and had very high learning rates (model 1: learning rate = 0.04) or low batch sizes (model 2: batch size = 32).

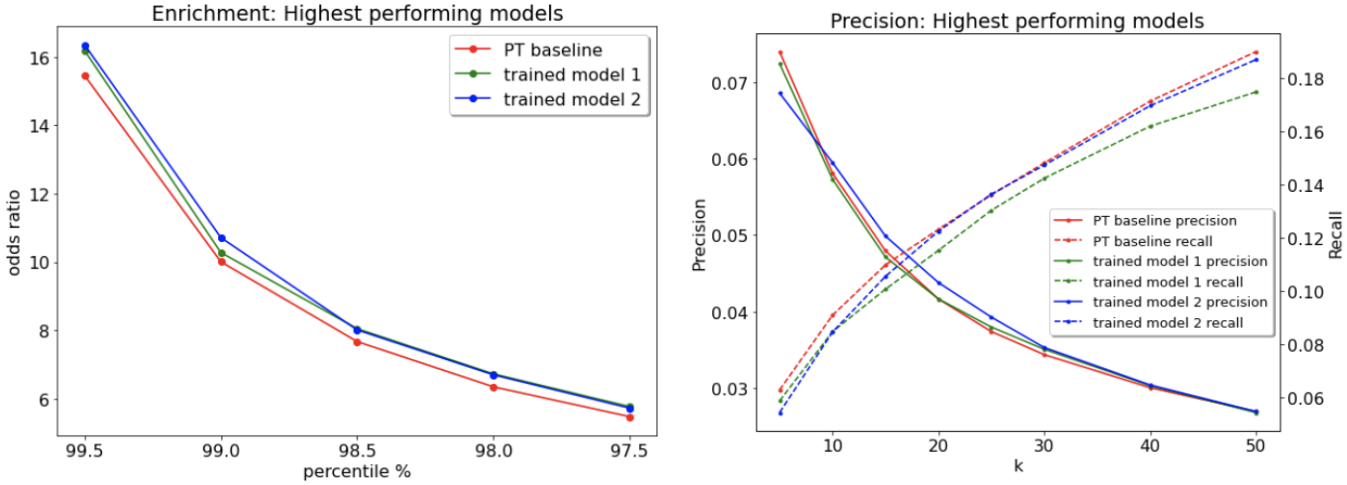

Figure 4.17.: *High performing models.* Experiment IDs: PT baseline, ID 200; model 1, ID 1028; model 2, ID 1103

Link to Discussion 5.7.8

4.8. Technical artifacts

If technical artifacts, such as batch effects and plate layout effects, are as strong or stronger than the biologically relevant features, extracting meaningful information from the data becomes a daunting or impossible task. In this section, I investigate the existence of technical artifacts in profiles from the three different extraction strategies.

4.8.1. PCA visualization

As mentioned before, technical artifacts can and will often outweigh the (biologically-relevant) phenotypical features of a cell. Hence, I expect PCA plots of the highest principal components to visualize the batch effects. Very similar to the section on feature space, I will present PCA plots in order to visualize batch effects. The following plots all show the first two PCs of Level 3 profiles (non normalized well-level profiles), colored by their batch number.

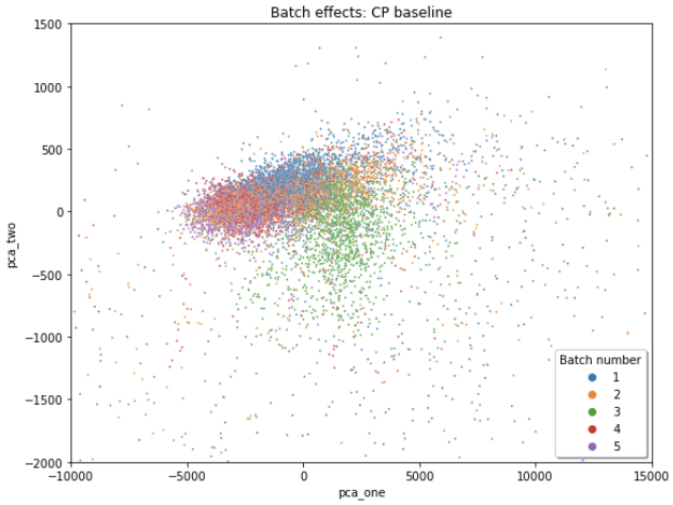

Figure 4.18.: *The effect of augmentation on training accuracy.* Experiment IDs: augmentation off, ID 1008; augmentation on, ID 1010

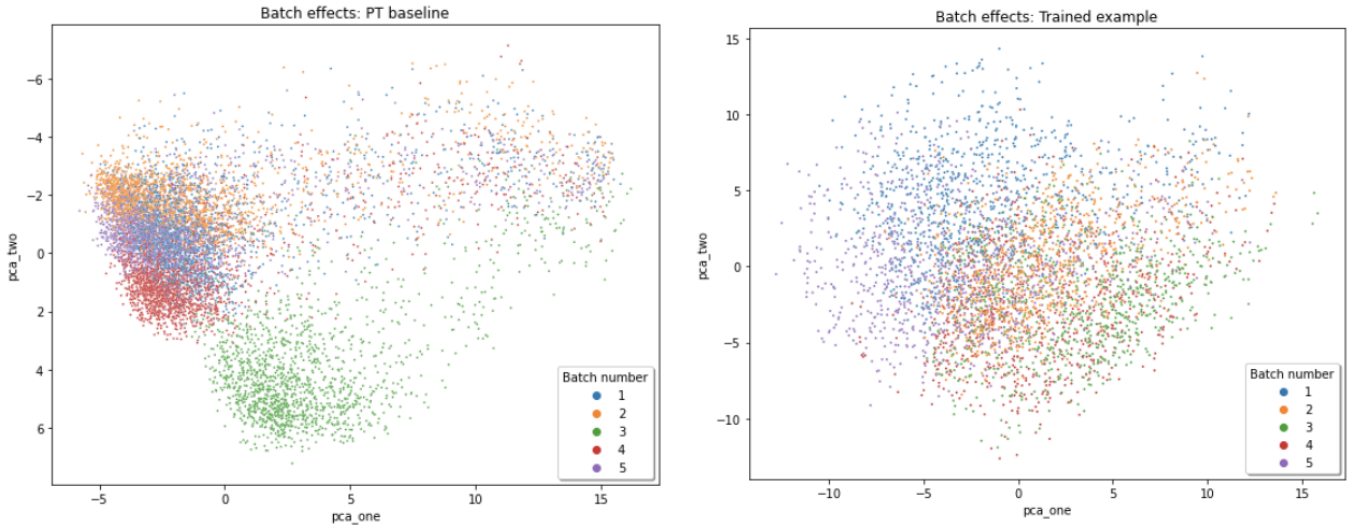

Figure 4.19.: *PCA plot showing batch effects.* a) CP baseline b) PT baseline c) Trained example . Experiment IDs: CP baseline ID 100; PT baseline ID 200; Trained example ID 813.

As a reminder, the explained variation per principal component for the first and second PC are CP baseline: 83%, and 8%; Pre-trained: 53% and 15%; Trained: 5% and 4%.

Of the three different scatter plots (4.18, 4.19), the trained feature space appears to have the smallest amount of batch effects visible and it also has the smallest explained variation, while in the pre-trained plot batch 2, 3 and 4 overlap very little. The PCA plot of the CP output appears to have a medium amount of overlap. The PCA plot for the trained features is chosen as a typical example from the long list of trained profiles I have acquired. Most plots look very similar with similar explained variation, despite the differences in training parameters and training data. Even when starting with a random initialization of the model, the resulting PCA plot is comparable to the ImageNet embedding initialization (see *Appendix* 6).

Link to Discussion 5.8.1

4.8.2. DMSO batch correlation

In an attempt to quantify the batch effects present in the well-level data, I have chosen to measure the *Batch enrichment score* (BES) of DMSO wells. Put simply, this metric will score high if DMSO well profiles of the same batch are more similar than those of different batches. I compute the *Enrichment* score over all 3,264 negative control wells with the batch number for the replicate groups (see *Cytominer-eval* 3.4) - marking wells from the same batch as replicates. This is very similar to the *Enrichment* score which uses MOA classes for the replicate groups. Reporting the BES at the 99.5% mark will allow an indication of the batch effects present in this data.

Augmentation decreases batch effects
Table 4.19 exhibits some key profiles with their *Enrichment* performance values (before and after spherizing) and BES (before and after spherizing). The *batch enrichment scores* reach very high values and exhibit a high variance across the models - even between models that are very similar. Despite this making comparisons difficult, I found that models trained with augmentation show lower *batch enrichment scores* than their comparable counterparts.

Pre-trained model shows lowest batch effect
Interestingly, the pre-trained EfficientNet model displays the lowest *batch enrichment scores* and applying a spherizing operation, fully cancels any batch correlation. For trained models with very high batch effects, spherizing reduces but does not fully diminish the BES. Unfortunately, no relationship between the raw *Enrichment* scores and the batch correlation scores can be extracted from this data.

Name	Perf. before spherizing	Perf. after spherizing	BES before spherizing	BES after spherizing
Low perf. model, augmentation off	3.64	9.76	23,097	72
Low perf. model, augmentation on	3.60	12.89	3,277	63
High perf. model, augmentation off	2.61	15.11	11,493	61
High perf. model, augmentation on	4.0	16.17	4,588	55
PT baseline	1.94	15.45	1,056	0.9

Table 4.19.: Comparing the batch enrichment score. performance scores refer to the 99.5% odds ratio of *Enrichment* performance. Experiment IDs: Low perf. model, augmentation off, ID 1008; Low perf. model, augmentation on, ID 1010; High perf. model, augmentation off, ID 1028 ; High perf. model, augmentation on, ID 1101. perf.: performance; BES: Batch enrichment score

Link to Discussion 5.8.2

4.8.3. Validation accuracy

In this subsection, I discuss how the selection of training data and validation data (e.g. Top50 and Strong Phenotypes) influences the validation accuracy and what this tells us about the technical artifacts.

Reiterate batch distribution

Let us briefly review the way compounds are spread across batches in the data. Four of the five technical replicate (of each compound/dosage) wells were produced in the same batch. We can select the training cells (the cells we train on) to contain cells from all five wells or from a few selected wells. These differences have a clear impact on the validation accuracy.

Data-split one

In the first experiment I ran, I randomly selected two training wells and one validation well from the five which were available. Hence it was often the case that the model trained on cells from one batch (called batch X) and then validated on cells from a different batch (called batch Y). This leads to a considerably lower validation accuracy, as technical artifacts make cells from batch Y very hard for the model to classify. I call this setup *data-split one*.

Data-split two and three

To solve this problem, I made sure that technical replicates three to five were used for training and cells from technical replicate one used for validation. In other words, in this second data split, *data-split two*, two wells from batch X and one well from batch Y were used for training and one well from batch X for validation. In my third data split, I decided to ignore wells and their batch membership and just randomly sample cells from all five wells so that both training and validation included cells that originated from all five wells. This is not ideal from a machine-learning perspective, because we train on all wells and do not hold out any validation data, but was useful in its ability to show the batch effects. Figure 4.20 shows how the validation accuracy for *data-split three* increases further because the model has trained on the same wells it validates on. The final validation accuracies can be found in Table 4.20. The smallest accuracy gap (0.02) of all experiment occurred during an experiment with a large training set (2 million cells) of type *data-split three* with augmentation (ID 1102).

In conclusion, the validation accuracy exemplifies how batch effects influence the classifiers to performance. Ignoring the fact that we are training and validating on the same wells, we have managed to fully diminish the accuracy gap and thus the model overfitting.

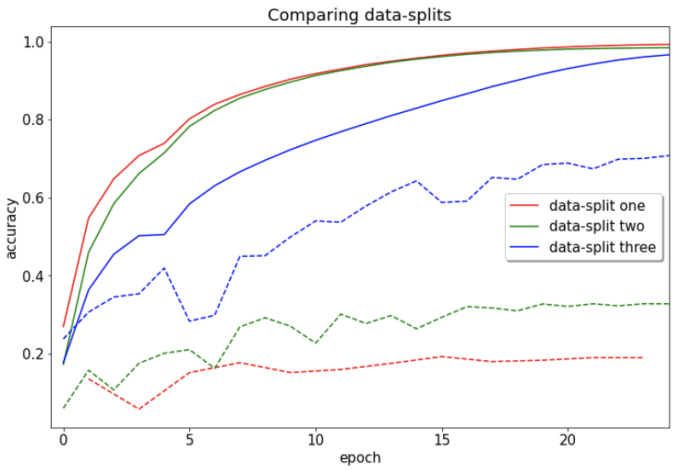

Figure 4.20.: Comparing validation accuracy for different data splits. Experiment IDs: data-split one, ID 813; data-split two, ID 827; data-split three, ID 1021.

Link to Discussion 5.8.3

Name	Training accuracy	Validation accuracy
Data-split one	0.991	0.189
Data-split two	0.983	0.327
Data-split three	0.977	0.723
Smallest accuracy gap	0.967	0.946
PT baseline	1.94	15.45

Table 4.20.: *Comparing validation accuracy for different data splits.* Experiment IDs: *data-split one*, ID 813; *data-split two*, ID 827; *data-split three*, ID 1021; smallest accuracy gap, ID 1102.

4.9. Pipeline, Resources and Speeds

Part of my research goal was to determine the computational resources needed to move data, sample cells, train models, and infer profiles. Over the course of the project, I have therefore tested different resources and tracked how much time is required to run certain aspects of the pipeline. This section will list my findings and will inform partners of the JUMP Consortium which resources to employ in the future. In the following paragraphs, my estimates are based not on the full LINCS dataset, but on my subset, which is around one-fifth of the entire LINCS set. The subsection consists of 8,700 wells with 18 million cells.

Exporting cells

Exporting single-cell crops requires the machine to open and close an image, crop it, and then save the crop. It is impossible to parallelize this process unless the computer has several I/O accesses to the disk. The JUMP project might need to use several machines that crop images at the same time and add them back together. However, this is clumsy and not intended by DeepProfiler. For my data, exporting the entire 18 million cells took just under eight days (175 hours). That is equivalent to 100,000 cells per hour. I did not gather evidence on what types of machines are faster or slower at exporting, but I assume it will depend on the processing power and more importantly, the access speed to the disk.

Training

The training speed depends on the RAM, GPU computing power, and GPU memory. RAM is relevant because DP holds the validation cells in RAM and the GPU memory is needed for holding the cells of each batch. Naturally, the number of validation cells is limited by the machine's RAM and the batch size is limited by the GPU's memory.

Training speeds also depend on training parameters such as augmentation, batch size, and training set size (how many cells are in training).

CHTC server (NVIDIA Ampere 100 SXM4) trains

$\sim$450 cells/second for a 256 batch size
$\sim$350 cells/second for a 64 batch size

The P3 GPU (NVIDIA Tesla V100) from EC2 trains 2-3 times slower:

$\sim$ 120 cells/second for a 64 batch size

Putting the above speeds into context: Training 30 epochs with a 256 batch size on a training sample of 0.4 million cells takes 11 hours. Turning on augmentation will increase training time by a factor of 1.3-1.5. Training on slower machines, e.g., the P2 GPU (NVIDIA Tesla K80), will further increase the training time by a factor of five. These calculations make it clear that if the JUMP project trains models, it seems best to create a 1-3 million cell training batch, which will lead to training times of between 25 and 50 hours.

Profiling

Profiling speeds depend on the GPU computing power, the batch size, and GPU memory size. In the context of profiling, the batch size is the number of samples that are passed forward at once. For the CTHC 40GB GPU, I found a batch size of 512 to be the most efficient. Note that this refers to profiling batch sizes and not training batches. Higher values may lead to memory leaks. The profiling operation of DeepProfiler works on one site after another (a site is a single multi-color image representing one field of view) and reports these in the standard output.

Macbook CPU:
$\sim$ 100 seconds per site

P2 GPU (NVIDIA Tesla K80):
$\sim$ 10 seconds per site

The **P3 GPU** (NVIDIA Tesla V100):
$\sim$ 2.5 seconds per site

CTHC server (NVIDIA Ampere 100 SXM4):
$\sim$ 0.25 seconds per site

In the context of profiling my LINCS data (78858 sites), profiling on the CHTC GPU is a six-hour operation. This is equivalent to 15 minutes per plate. If a higher throughput is ever needed, several profiling operations could be run on the same GPU or several GPUs can simply infer the features of several plates at the same time.

Cost comparison

CellProfiler runs on large farms of CPUs. The cost for extracting profiles runs at $\sim$5 CPU minutes per site resulting in 288 CPU hours per plate, equivalent to around $10 on EC2 servers [44]. The cost of running 8 NVIDIA A100 GPUs on EC2 is around $10 per hour. Taking into account that we can infer 4 plates per hour on one A100 GPU, my best estimation of the cost of profiling one plate stands at approximately $0.4 per plate.

Downstream pipeline

The downstream pipeline steps are considerably faster than the training and inference steps. Running the well aggregation with Pycytominer.DeepProfiler_Processing normalized accumulated scores of the histogram for the LINCS dataset used in this thesis takes about an hour on the CTHC server. Loading, spherizing and evaluating with Pycytominer and Cytominer-eval takes only a few minutes on a Mac book pro.

Storage space

Storage space is not often the cause of bottlenecks in these projects. However, it is worth noting that the input folder's size of the DeepProfiler project folder is 900 GB. This folder contains all images and location files. The single-cell crops file containing all crops is ∼600GB and profile output folders store circa 80GB.

5. Discussion

This chapter will follow the same structure as the *Experiments and Results* chapter, making relations and references between the two chapters easier to understand and follow. I will also include links to the relevant experimental data at the beginning of each section.

5.1. Experimental design and data Selection

Link to Experiments and Results 4.1
Selecting a subset of the complete LINCS database for my experiments was a critical decision because it allowed me to train models, profile cells, run the downstream pipeline, and calculate evaluation metrics at a reasonable pace. By making sure the data applies to all metrics without further modification (e.g., only keep MOAs with MOA size larger than two), I was able to run the downstream processing and performance calculations in one quick script. In hindsight, it was also beneficial not to cut the number of compounds, since I show that different subsets of data perform very differently and a diverse LINCS dataset is most representative of the JUMP data. In future work, it would be beneficial to train on one dose and then evaluate metrics on a second dose. This cross-dose evaluation realistically simulates new data and allows a better understanding of when the model is overfitting to the training data.

5.2. Metric Comparison

Link to Experiments and Results 4.2
Agreement between two evaluation metrics lends confidence in choosing the best-performing profiles and, by extension, the model with the best-extracted profiles. For example, if *Enrichment* shows that model A is better than B is better than C, *Precision@K* agrees with *Enrichment* if it shows the same ranking. Note that I will use *the performance of a model* interchangeably with *the performance of profiles* extracted from the same model using DeepProfiler.

Using several metrics is beneficial
It is the correct choice to apply different evaluation methods to the data instead of only one metric. While using only one metric makes the comparison of profiles easier and faster, the multi-metric approach is helpful at several points in the analysis. Often the *Precision@K* metric shows very little difference between two profiles, but then *Enrichment* allows me to differentiate them and vice versa.

Hit@K was not reported
When comparing *Hit@K* to the other three metrics (*Enrichment*, *Precision@K*, Recall@K), *Hit@K* often does not clearly differentiate the profiles (when other metrics did) or disagrees with them on the best

performing model. Hoping to better compare different profiles, I developed the representation with normalized accumulated scores (see Figure 3.3) which act like the cumulative density function of the original *Hit@K* histogram. While this metric may be an intuitive tool to show to other scientists and mimic the application behavior more transparently than the other metrics, I found it less helpful in comparing many different profiles. Hence, *Hit@K* is calculated for all experiments but rarely mentioned in the *Experiments and Results* chapter and not taken into account like the other metric scores.

Precision@R is reliable
In contrast to *Hit@K*, *Precision@R* proves to be a reliable method. Almost all *Precision@R* values agree with the ranking of *Precision@K* and deliver a single number alternative to the *Precision@K* plots. The value of *Precision@R* is often around the same value as *Precision@K* for $K = 5$, which is unsurprising given the fact that most MOAs have a class size between two and five.

In conclusion, reporting the *Enrichment* score (odds ratio) at 99.5% and the *Precision@R* score is sufficient to predict the profile ranking that becomes more apparent with the complete set of evidence.

5.3. Downstream Pipeline

Challenges of CP profiles 4.3.1
The downstream pipeline for CellProfiler data is a complex process depending on several parameters. The challenges of CP data - highly-correlated features and non-normalized data (feature values can range from -100 to 100) - can only be met with sophisticated feature selection and normalization schemes. Unsurprisingly, I found the best performance by adding a final feature-selecting operation on Level 5 data. Despite these challenges, the CP baseline scores are higher than some profiles stemming from the trained strategy.

Pipeline for pre-trained profiles 4.3.2
As expected, spherizing is by far the best operation to increase performance. It improves the *Enrichment* metric for the profiles from the pre-trained strategy (pre-trained profiles) by a factor of eight, while all other normalization techniques only improve *Enrichment* performance by a factor of three. Reminding ourselves that the feature space of convolutional networks is scaled with batch normalization layers that result in similar value ranges, it becomes clear why simpler normalization techniques (like *standardize*) have less of an impact on the evaluation metrics. Contrary to CellProfiler data, which requires a prior normalization operation, pre-trained profiles show the same performance by only spherizing (without any normalization). I confirm that feature selection only minimally impacts the performance scores, which indicates that the pre-trained feature spaces are less correlated than the CP feature spaces. However, the above argument on normalization and uncorrelated features only applies to a certain extent, as the pre-trained model is not trained on cells but instead natural images. Thus, the vector for representing cells is not necessarily near the origin.

Pipeline for trained profiles
The different pipelines for trained profiles perform similarly to the pre-trained profiles. However, the feature selection now has a negative impact on performance. One could argue that the dimensionality of the two feature spaces (6000 pre-trained vs. 1280 trained) explains this difference. The lower dimensionality of trained profiles increases the density of information per feature. Hence, dropping a large number of features is unlikely to improve performance. Moreover, the odds ratio for the raw profiles (no

normalization) is considerably higher for the trained strategy, which may point to even more normalized, uncorrelated, and zero-centered features.

In conclusion, the results of the pipeline analysis confirm that pre-trained nets can outperform classical feature extraction. We have further gained evidence that deep learning profiles are, to some extent, already normalized and uncorrelated. And finally, we choose a simplified downstream pipeline and only perform spherizing on pre-trained and trained profiles. Note that I am applying operations that were developed for CellProfiler data to pre-trained and trained profiles. It seems likely that further research into the optimal downstream pipeline for DeepProfiler profiles will improve scores.

5.4. Feature Space

Link to Experiments and Results 4.4
The analysis of the PCA plots and explained variation further confirms the conclusions about feature correlation from the above section. Via the explained variation of the first two principal components (PC), I show that feature correlation is the highest in CP and the lowest in trained profiles. The non-centeredness of pre-trained profiles likely leads to a higher explained variance compared to trained profiles. Interestingly, both the CP and pre-trained PCA show many wells visibly outside of the domain of DMSO wells. This is not the case in the trained profile PCA plot; however, there may be a clear distinction between DMSO and non-DMSO wells in a different PC. An open question is whether the larger feature space of the pre-trained extraction strategy affects the PCA or the explained variation.

5.5. Baseline Performance – Pretrained and CellProfiler

Link to Experiments and Results 4.5
All evaluation metrics agree on the fact that the pre-trained profiles outperform the CellProfiler baseline. This is the central finding of this thesis and a valuable reaffirmation that deep learning models have many advantages over classical methods, the two most important being higher performance and more straightforward downstream processing.

The ResNet and EfficientNet performance are relatively similar, indicating that the architecture of pre-trained networks does not significantly impact the ability to create helpful feature representations. However, some ResNet features have very low variation creating a numerical failure during spherizing unless feature selection is performed prior to spherizing. The same issue occurs with trained ResNets (with 2,048-dimensional profiles), indicating an architectural issue. Subsequent research is needed to investigate the exact origin of this issue.

5.6. Baseline Performance – Subsets

Link to Experiments and Results 4.6
I created the Top50 subset to test whether compounds from large MOA classes perform better on *Precision@K*. The in-Top50 *Precision@K* is $\sim$1.7 times higher than the standard (full data) *Precision@K*

value and four times higher than the out-Top50 score. This confirms the concerns around *Precision@K* being positively biased towards large MOA class sizes. Unexpectedly, *Recall@K* and *Enrichment* also increase performance for the in-Top50 set. The reason for this increase might be that compounds in-Top50 have more dramatic phenotypes and "stronger" profiles even though there is no evidence for substantial overlap of compounds with the Strong Phenotype dataset (225 out of 520). Confirming my mathematical prediction of high precision scores due to large MOA sizes, the Precision@K values for the in-Top50 set are higher than those of the in-SP (or any other profiles).

The extremely high performance of in-SP data shows the significant inhomogeneity in the dataset regarding performance on a compound level. While the *Enrichment* score increases by a factor of two compared to the full data score, the *Precision@K* only increases by 1.25. This further confirms the observation that Enrichment is more sensitive to changes in the data than *Precision@K* and *Recall@K* are. Interestingly, *Hit@K* exhibits a similar increase factor (as *Enrichment*). This is one of the few experiments where *Hit@K* agrees with *Enrichment* scores.

5.7. Training Models

5.7.1. Learning rate and schedules

Link to Experiments and Results 4.7.1
The cosine learning rate of 0.02 achieves the highest validation accuracy and performance scores. Flat learning schedules are generally not recommended, because they can lead to unstable convergence at later epochs. On the other hand, the cosine schedule exhibits very small learning rate values (< 0.001) after more than 20 epochs. Although I did not improve the cosine learning schedule with custom step functions, there may still be potential for improvement. Overly-high learning rates can be identified by diminishing accuracy (and loss) improvement. Figure 4.9 (see red curve) exhibit a "step" of the training accuracy at the fifth epoch which occurs due to the highest learning rates of the cosine learning schedule occurring at this point (also see *Appendix* 6).

5.7.2. Batch sizes

Link to Experiments and Results 4.7.2
The outcome of these experiments is quite clear and in line with the expected effects of batch sizes. While training on smaller batches increases training time, it has a strong positive effect on performance. I further confirm the well-documented interaction between batch size and the learning rate. Testing more extreme batch sizes - below 32 and above 256 - remains a task for future research.

5.7.3. Epochs

Higher performance for higher epochs 4.7.3
Training longer increases the performance scores. Due to time constraints, I did not measure the performance at later points in the training where training accuracy has seemingly converged and I did

not reach the convergence of the performance values. Hopefully, further experiments will find the point where performance does not increase anymore so that a good recommendation for epochs can be made. Currently, my best estimate for training data of 0.5 - 2 million cells is to use approximately 30 epochs for training.

Performance decreases before increasing
Interestingly, initializing weights with a pre-trained model and training for only a few epochs strongly decreases the performance (compared to the pre-trained starting state). However, initializing the weights randomly (see *Appendix* 6) leads to even lower evaluation metrics. This implies that - to some extent - I am "rewiring" the model such that the feature representations of the pre-trained CNN are partially, but not entirely, altered. Future research could focus on adapting our weakly-supervised approach to preserve the pre-trained structures. Ideally, the model's performance would improve after each epoch instead of early epochs having a low performance until the model converges to a well-trained cell classifier. Freezing the weights from early layers (at the front of the CNN) could potentially preserve the general feature abstraction ability of the neural net.

5.7.4. Label smoothing

Link to Experiments and Results 4.7.4
We introduced label smoothing (LS) into DeepProfiler as a regularization tool. Higher label smoothing values increased the loss function value, as expected (not shown in the *Experiments and Results* chapter). Interestingly, the training accuracy for LS=0.1 drops below 0.8, while the validation accuracy remained rather similar for all other LS values. Having the smallest gap between training and validation accuracy, LS=0.1 is succeeding at regularizing the model. Moreover, this model actually scored the best *Enrichment* values and performed average on *Precision@K* and *Recall@K*. I do not have any explanation for the drop in training accuracy of the LS=0.1 training experiment. Hopefully, a follow-up investigation will reveal more details on the effect of the label smoothing value on training and performance. With label smoothing showing less effect with higher-performing models (See *Augmentation and label smoothing* 4.7.5), I kept the LS values at zero for most experiments.

5.7.5. Augmentation

Link to Experiments and Results 4.7.5
The image augmentations performed during training do indeed regularize the model and decrease the overfitting gap (difference between validation and training accuracy). Moreover, the *Technical artifacts* section shows further evidence for diminishing batch effects through augmentation. The improved performance might stem from this reduction of technical artifacts in the data or from the fact that the model is exposed to a wider diversity of cells during training which improves the performance. However, augmentation does not always have a positive impact on the performance.

Augmentation and label smoothing

Subsequent experiments with variable augmentation and label smoothing indicate a more complex relationship than the previous section implied. For very high-performing models with high learning rates and low batch sizes, augmentation and label smoothing have less of an impact than for the lower-performing models. Assuming there exists an optimal feature space for this classification problem all models might be very "close" to this optimum, hence, the influence of the two regularization tools is minimal. Another potential explanation is that I have reached the maximum of performance that this data allows hence limiting the effect parameters have on the performance (see section *High performing models* 4.7.8). Subsequent research could investigate the performance improvement on larger training sets and how augmentation interacts with other parameters, such as label smoothing and learning rates.

5.7.6. Training on subsets

Link to Experiments and Results 4.7.6
Overall, training on subsets of data has allowed me to perform many experiments while also uncovering interesting observations about how training influences performance. Training on subsets and then evaluating metrics on untrained data also acts as a proxy for unknown data that may be added in the future.

Top50
As already shown in the *epoch* section, we can observe the improvement of some evaluation metrics through training. Both training on the Top50 and the SP set improves the in-set *Enrichment* scores considerably. This confirms the hypothesis that training on a subset X will increase the performance of that subset X and potentially decrease the performance of the out-subset. However, training on the Top50 set does not affect *Precision@K* and *Recall@K* scores of the in and out-Top50 set. Note that we already knew from the *epochs* section that the *Precision@K* and *Recall@K* scores did not increase for models trained on the full data.

Strong phenotype
For the strong phenotype set, we observe a similar story. Training a model on SP cells increases the validation accuracy, the *Enrichment*, and the *Recall@K* scores of the in-SP compounds while decreasing the out-SP scores. The gap between in-SP scores and out-SP scores increases considerably compared to the pre-trained in/out-SP gap. This implies that learning a model on the SP data decreases the ability to represent the low phenotype (out-SP) data compared to the pre-trained model. Contrary to the Top50 training, *Recall@K* now exhibits a strong in/out-SP gap, but *Precision@K* still is not affected by the data split. The validation accuracy increases by a factor of 1.3, further confirming that the SP set contains high phenotype compounds, which appear easier to classify when compared to the Top50 set or all compounds. Overall, we know that *Precision@K* and *Recall@K* are less "sensitive" to feature space improvements driven by training, but I have no explanation for why *Precision@K* (in both cases) and *Recall@K* (in the case of Top50) show so little difference.

out-SP performance
Finally, I showed that the out-SP performance weakly depends on whether the model has trained on the entire data or only the SP subset. This is interesting, because it means that a model trained on all phenotypes is marginally better at representing low-phenotype cells than a model trained on only

strong phenotypes. Nonetheless, the overall performance of the model trained on all compounds slightly outperforms the SP trained model.

There is no easy conclusion or recommendation to be gleaned from this data. We learned that cells with strong phenotypes train and perform considerably better than cells with weak or mixed phenotypes. We further learned that training on weak phenotype cells only marginally increases their representation performance, whereas the opposite is true for strong phenotype cells (increasing the in/out gap). Finally, we also show that training on the full dataset is marginally better than training on the SP subset of compounds. This rules out the idea that better models will arise from removing low-phenotype cells, indicating that these cells do not just add noise to the model. I will elaborate on this discussion in the next section.

Future investigations could train a model on the out-SP set and see if its performance decreases the in/out SP gap. Finally, it would be interesting to learn why the *Precision@K* score is not affected by the subset training experiments.

Training on all compounds

Link to Experiments and Results 4.7.6
This section details a comparison of two experiments with different numbers of compounds in the training set. Following a trend observed in many different experiments, the comparison shows how performance generally increases when the diversity of the training set is larger. The same effect can be shown for training sets that use only a subset of batches (lower performance) instead of all batches. Therefore, I decided to train the high-performing models on a random selection of cells from all compounds and all batches, even though this inhibits me from validating the model on unseen wells or batches.

5.7.7. Training on larger datasets

Link to Experiments and Results 4.7.7
Unsurprisingly, larger datasets lead to better models. The performance differential between one and two million cells is not marginal, implying that further training size increases will enhance the performance. However, training on more than two million cells takes at least three days, so this becomes a resource constraint. Keeping in mind that the larger training experiments run longer and perform more gradient descent steps in total, it is not surprising that the larger datasets show better results. In conclusion, these experiments give additional evidence that trained models may be able to outperform pre-trained networks when trained on large datasets.

5.7.8. High performing models

Link to Experiments and Results 4.7.8
The highest performance models further show that some hyperparameters are less important than others when solely optimizing for performance. The most important parameters are a high learning rate, a low batch size, high epoch numbers and large training sets. The last two parameters linearly increase the training time, limiting the number of experiments performed in a short period of time.

It is conceivable that I am close to the maximum performance possible in this dataset, which would explain why I did not succeed in train models that outperform the pre-trained model. On the other hand, all experiments have run less than 100 hours on the A100 GPU and longer training times may result in considerably stronger models.

5.8. Technical artifacts

5.8.1. PCA plots

Link to Experiments and Results 4.8.1
The PCA batch plots in this section confirm that batch effects dominate the high variation principal components, even in the trained strategy case. Visually, the trained feature space appears to have the largest overlap of batches. When taking the results from the next section into account, it seems that this visual inspection gives few insights into the technical artifacts. Future research could visualize the well position within the PCA plots to uncover potential well-position effects.

5.8.2. DMSO batch correlation

Link to Experiments and Results 4.8.2
Batch enrichment scores are not a well-established method and I am thus unsure how well the score measures the overall batch effects present in the data. For now, it is the best approximation available.

Augmentation regularizes models
The results from these experiments confirm our hypothesis that image augmentation regularizes the trained models. Despite the high variance of batch enrichment scores, this trend is valid for most pairs of augmented and non-augmented models. The pre-trained baseline, however, achieves by far the lowest *batch enrichment scores* of all non-spheriezed profiles. This implies that trained models encode more batch effects than the pre-trained models, a statement that contradicts our assumptions and evidence from previous experiments (including the explained variation in the PCA plots and the higher raw performance).

Spherizing deminises batch effects
Moreover, we learn that the batch effects we are measuring with the *batch enrichment scores* do not determine how well profiles perform, as the *batch enrichment scores* of the high-performing models are two magnitudes higher than the PT baseline but still outperform the baseline. Interestingly, higher performance for trained models does correspond to lower *batch enrichment values* for the spherized profiles. We also learn that the spherizing operation is not able to fully diminish the batch effects from trained models. This is a key learning because it encourages us to further seek methods to combat technical artifacts.

In future work, it would be very useful to gain a better understanding of what the *batch enrichment scores* measure and how this affects the performance. Applying different metrics that measure technical artifacts will help understand the full picture. The same analysis done here for augmentation should also be repeated with the other regularization methods (label smoothing and L2-regularization) to gain an

understanding of which method has the biggest impact. My initial hypothesis was that training on several batches and applying regularization techniques would make models robust against technical artifacts. However, these results seem to indicate that the opposite is occurring. Whether the hypothesis is incorrect and how to further avoid learning batch effects should be the focus of subsequent research.

5.8.3. Validation accuracy

Link to Experiments and Results 4.8.3
First and foremost, it is evident that most models overfit to a rather large degree. The gap between training and validation accuracy is as large as 0.8, depending on the relationship between training and validation cells, epochs, and a number of other training parameters.

Learning well positions
The question is whether the model correctly classifies cells based on their phenotype or has learned to identify wells and plates and thus implicitly classifies compounds correctly. The gain in validation accuracy (VA) from *data-split one* to *data-split two* already hints at the latter assumption. When moving to *data-split three*, the VA increases by a factor of two once more. Unfortunately, I have not overcome the overfitting issue in this third case because I am training and validating with cells from the same wells. Nonetheless, it makes sense to include all wells into the training data, as my goal is to improve overall performance.

A core issue with the LINCS dataset is that technical replicates are found in the same well position on different plates. Well position effects can lead compounds to be identified as their well and less as their phenotype. A further investigation could focus on classifying data from outside the LINCS subset, for example, from a different dosage. This strategy will result in more realistic estimates of performance and also allow for a better understanding of the technical artifacts learned in a model. Potentially, new datasets will be designed without compounds being set onto the same wells and (mostly) onto the same batch. Hopefully, this would solve the issue of learning well positions instead of compound attributes.

Ultimately, the classification of wells into compounds is an auxiliary task, and while overfitting is a problem in general, it does not seem to affect our performance metrics on the primary task of classifying compounds into MOA. For now, this statement only holds true when training occurs on the same data that the evaluation metrics are performed on.

5.9. Deciding between training and pre-trained models

Trained and pre-trained models are on par
I have investigated a wide range of hyperparameters and subsets of data to optimize the performance of trained models. I was able to train excellent compound classifiers with validation accuracies up to 65% but I was not able to outperform pre-trained models on the evaluation metrics. However, some of the parameters I investigated show the potential to further improve the trained models if given enough resources. I believe we will be able to clearly outperform the pre-trained data by increasing training set size and the number of epochs, using lower batch sizes, and further optimizing the learning rate schedule, label smoothing, and regularization parameters. I have also shown clear evidence that training

on sufficiently diverse data can considerably improve performance on that set of data. Hopefully, this implies that - with larger models and longer training time - even the performance of less phenotypically strong data can be increased. Nonetheless, it is impressive how pre-trained models that have never "seen" cell images have the ability to represent cell features and to achieve very high evaluation metric scores.

5.10. Generalizing to other data

Learning applicable to other datasets
Future research may want to apply pre-trained and trained extraction strategies to other datasets. It is important to consider which methods and learning from these experiments are applicable to other data. Certain learnings and conclusions are most likely applicable to all datasets because they are more general statements of machine learning. The effects of batch sizes, learning rates, and higher epoch numbers fall into this category. Similarly, training on larger datasets and training on the full range of batches (higher diversity) will be likely to improve model performance for any dataset.

I do not know if there is anything special about the LINCS data that makes training on it more or less successful than other data. I would expect smaller datasets to benefit more from image augmentation and label smoothing than a large set with inherent diversity. As long as the data quality is sufficient, I am rather confident that deep learning-based methods will always outperform CellProfiler profiles.

Moreover, the downstream pipeline should be easily transferable to other projects. Spherizing has proven to be the best downstream normalization and batch correction method on the different types of profiles (CP, pre-trained, and trained extraction strategies), hence it would be surprising to see spherizing not being very helpful. The broad application of *Enrichment*, *Precision@K*, and *Recall@K* in this project promises that these evaluation metrics will be useful to other projects as well.

5.11. Recommendations for JUMP

Because this project is aimed at preparing a pipeline for the JUMP project, I have included this section of recommendations. I will summarize the most important learnings from my thesis and conclude with a list of topics to further investigate.

LINCS subset
When comparing my performance results to other results, one must apply the same data selection as I did. The LINCS subset used throughout this thesis is described in the *Methods and Materials* chapter 3 and my repository. Below, I discuss how to expand on this data to allow for better training experiments.

DeepProfiler
The DeepProfiler codebase has developed considerably during the last months due to its application in this project. It has proven to be a reliable and efficient method to train models in a weakly-supervised fashion and infer profiles from pre-trained or trained models. My documentation, including scripts, docker containers, and practical instructions - as detailed below - should allow all users to efficiently perform their training and profiling.

Extraction pipeline

The profile extraction process requires loading the images and location files onto the GPU resource and running DeepProfiler on a Docker container to extract features. Next, Pycytominer's functions DeepProfiler Processing, Spherizing, and Aggregation are used as downstream processing steps to acquire Level 5 consensus profiles. With access to powerful GPUs such as NVIDIA Ampere 100 SXM4 GPU, profiles can be extracted at a reasonable speed. The LINCS data used in my project (ca. 25 plates) can be profiled and processed within approximately six hours on one A100 GPU (30 hours for the entire LINCS dataset with 136 plates). These time estimates do not factor in the time required for preprocessing images (10 CPU hours/plate) and generating cell locations (300 CPU hours/plate using CellProfiler, although this step will be performed anyway for the JUMP dataset) both processes that are easily parallelizable to run 100 times faster. Models that are on par with the pre-trained EfficientNet can be trained within a few days. The *Pipeline, Resources, and Speeds* section 4.9 further details all resource requirements and rates of profiling and training.

Methods

I recommend always applying the Pycytominer spherizing function to well-level data, as it creates an orthogonal features space and diminishes batch effects, thereby generating the best performing data. Results indicate that batch effects are fully diminished in profiles from pre-trained nets while being largely diminished in profiles from trained models. Furthermore, I have gathered ample evidence that the Cytominer-eval functions *Enrichment*, *Precision@K*, and *Recall@K* are very useful metrics to measure the performance of a given profile feature space. I recommend following or using my evaluation scripts when performing downstream processing and evaluation metric analysis.

Use pre-trained over CP

I have found a range of reasons to use pre-trained nets over CellProfiler for feature extraction. Most importantly, the EfficientNetB0 architecture, pre-trained on the ImageNet dataset, shows superior performance over CellProfiler. Secondly, pre-trained feature space is generally less correlated and more normalized, allowing us to drop all downstream processing steps apart from spherizing. From an infrastructural perspective, applying DeepProfiler for profiling is also considerably faster and cheaper than using CellProfiler by a factor of approximately 25. Similar to Distributed CellProfiler, a distributed and automated wrapper around DP can and probably should be developed which would allow JUMP partners and other users to easily infer deep learning features.

Trained models do not outperform pre-trained, yet

I have investigated a wide range of hyperparameters and subsets of data to optimize the performance of models trained on cell images. I succeeded in training excellent classifiers with validation accuracies up to 60% but was unable to clearly outperform pre-trained CNNs. If fast and easy access to a model is a priority, and no further research is committed to optimizing trained models, I recommend employing the pre-trained EfficientNetB0 model to extract profiles.

Potential for trained models

However, my experiments show clear potential to improve the trained models further if optimized and given enough resources. I believe we will be able to outperform the pre-trained data by increasing the size and diversity of the training set size, increasing the number of epochs, using smaller batch sizes, and further optimizing the learning rate schedule, label smoothing, and regularization parameters. I recommend that the JUMP consortium further investigate the improvement of trained models and build upon my thesis' experiments, learning, and recommendations. I also recommend investing in research on further combating technical artifacts with deep learning and creating metrics that quantify the technical

artifacts of a given dataset.

I have gathered all further investigation topics in the following section and call upon readers to follow up on these open questions.

5.12. Further investigations

I recommend investing resources into the following topics because they promise to increase the performance of trained models and give additional insights into technical artifacts and how to combat them.

Open hyperparameter questions
First, I recommend following up on a list of open questions stemming directly from the experiments I performed. The L2 weight decay parameter (currently set to at 0.0001) can be investigated for its potential to decrease the model from overfitting and potentially increase performance (similar to augmentation). Additionally, I have not created enough evidence to definitively characterize the effect of label smoothing on the performance, nor have I investigated the impact of very small batch sizes (<32). With more time and computational resources, models can be trained on more cells (more than two million) while keeping batch sizes low and epoch numbers high. The goal (apart from higher performance) will be to find the point at which performance converges and does not further increase for higher epochs. Additionally, questions around the training of subsets still exist, e.g., how training on an out-SP data set will affect performance or why *Precision@K* does not increase when other metrics do increase.

Use additional LINCS data
A significant addition to this line of experiments would be incorporating unused LINCS data, specifically, other doses. The second-highest dose (dose number five) has been shown to create equally strong phenotypes in cells. This fifth dose can be used for training data, validation, or analysis. If we use the fifth dose as training data (in addition to the current dose six), the model would be exposed to two different wells for the same compound and thus become more robust regarding well-position artifacts. We could further use the fifth dose as validation data (instead of dose six) and receive a more accurate understanding of the model's performance (and level of overfitting) as we would not train and test on the same data anymore. Finally, we could use the extra data as our evaluation data. Instead of training on X and then evaluating X, we could train on X and then evaluate on profile Y, thus testing the models' ability to create high-performing profiles on unknown data. These three options (which are combinable) will considerably increase our understanding of the performance and the overfitting of the trained profiling strategy. I consider this to be the most important next step in continuing my research.

Analyse batch effects
I further recommend the JUMP consortium and the image-based profiling community at large to extend and formalize the tools to measure technical artifacts. I have used the *Enrichment* scores and batch numbers as a first idea for measuring batch effects. Ando et al. [46]introduced the concept of "Not Same Compound or Batch" to factor in batch effects. Way et al. explored the effect of well position in a plate [42]. I recommend these ideas for identifying and quantifying batch effects to be fleshed out and formalized more.

Downstream processing

The members of the Carpenter-Singh lab have spent considerable time optimizing the downstream pipeline for CellProfiler profiles. This has culminated in the Pycytominer codebase. Similar efforts have been a part of other software projects e.g. EBImage and the recently released Julia library [51]. To my knowledge, little effort has been spent on optimizing the downstream processing for deep learning features. My thesis applies existing operations such as feature selection and normalization but does not test any novel methods. I think there is potential for additional batch effect correction operations - further increasing the performance of deep learning models.

Improving Hit@K

The Hit@K evaluation metric will need further improvement to be useful. The normalized accumulated representation that I chose as a representation metric was neither intuitive nor useful in comparing profiles. To improve this metric, we could focus only on the first 20% of the nearest neighbors, as this is the relevant section where pharmaceutical researchers would hope for good candidates. Moreover, one could develop a single value output that would make the best performing profiles clearly identifiable.

Single command line package

A current bottleneck for quickly profiling cells is acquiring the location files, which requires running the raw images through CellProfiler. In the future, a deep learning-based solution could be deployed to infer cell locations more efficiently. Creating a script that allows users to run DeepProfiler and Pycytominer in combination and over the same interface will also be advantageous. Currently, users need to understand the command line interface of DeepProfiler and the Python function calls of Pycytominer. Creating a standardized pipeline with a single function call for profiling, aggregation, spherizing, and evaluation metric analysis seems worthwhile.

6. Conclusion

In my master thesis, I developed a protocol for robustly and scalably quantifying (profile) cellular states from images using deep neural networks. The JUMP-Cell Painting consortium will be able to apply this protocol to profile over two billion cells in a dataset that will be publicly shared for the scientific community.

I successfully examined the potential of a novel profile extraction strategy - CNNs trained under weakly-supervised classification - and compared this method to CellProfiler. I found the main reasons for the use of deep learning-based (pre-trained and trained) strategy over CellProfiler to be: 1) superior performance, 2) a normalized and less correlated feature space, 3) simpler downstream processing pipeline, 4) and potentially less hands-on time. I have evaluated the three different strategies for profile extraction - CellProfiler, pre-trained, and trained CNNs - based on MOA profile similarity, i.e., the assumption that compounds with similar MOAs generate similar profiles.

The subset of data chosen for my thesis has been successfully applied to all analyses. It proved to be a subset small enough to perform processing operations and compute metrics while being large enough to train large CNNs on it. In addition to improving the DeepProfiler codebase, we have proven DP to be a reliable and efficient method to train models and infer profiles from pre-trained or self-trained models.

The optimal pipeline for CellProfiler data includes normalization, feature selection, and spherizing. For profiles from the trained and pre-trained strategy, we found that spherizing alone creates an orthogonal feature space, diminishes batch effects, and generates the best-performing profiles. Furthermore, we have gathered ample evidence that the Cytominer-eval functions *Enrichment*, *Precision@K*, *Precision@R*, and *Recall@K* are the most valuable metrics to measure the performance of a given profile feature space. I developed an additional evaluation metric *Hit@K* that offers an intuitive view of the data and mimics the application of MOA similarity search. All these functions have been made readily available through Github.

The self-trained profiling strategy includes training an EfficientNet with a single-cell classification model (using the cells' perturbation as class labels) as an auxiliary task and then using the internal representation space of the resulting model to profile cell images. I investigated the effect of several training hyperparameters on the performance of their resulting profiles. Optimal values or value ranges were found for the learning rate, batch size, label smoothing parameter, epoch number, augmentation, and for the number of cells that we train on. I was further able to show that trained models improve profile extraction for the cell space that the model was trained on. On the LINCS dataset, I was able to slightly outperform the profiles of the pre-trained strategy with my own trained models. Enough evidence is present to believe that further improvements to the training process will be able to clearly outperform pre-trained profiles.

Overall, I have prepared a clear strategy for the JUMP-Cell Painting Consortium to follow and recommended an extensive list of further research questions to investigate. I hope that my work will be instrumental in profiling the two billion cells of the JUMP data set in the months to come.

Glossary

Assay
A specific model system with biological relevance to the disease in question and an easily understandable readout is called an assay. The assay used to create all images in this project is called CellProfiler and is a low-cost profiling assay that uses six different fluorescent dyes.

LINCS
The LINCS database (short for Library of Integrated Network-Based Cellular Signatures) aims to create publicly available resources to help scientists characterize cells' phenotypic responses to perturbations. When mentioning LINCS, I am referring to the first batch found on the LINCS repository.

Cell Painting
The Cell Painting assay, which was developed at the Broad Institute in the Carpenter Lab, is a generalizable and broadly applicable assay for accessing the morphology of a cell state through a combination of six fluorescent dyes.

CellProfiler
CellProfiler refers to the software tool that is used to extract hundreds of measurements from millions of cells in a typical high-throughput imaging experiment.

DeepProfiler
DeepProfiler (DP) is a set of tools that allows machine learning scientists to train deep learning models and infer profiles from the cell images.

Pycytominer
Pycytominer is a suite of functions that post-process profiles from CellProfiler, DeepProfiler, or other profiling experiments.

Evaluation metrics
Sometimes also called quality metrics or evaluation methods, these metrics have been chosen to measure the usefulness of a given representation of our LINCS data to the pharmaceutical process. If a model produces profiles with high evaluation metrics, I consider this a model with high performance.

Cytominer-eval
Cytominer-eval contains functions to calculate quality metrics for perturbation profiling experiments. In this thesis, Precision-recall, Enrichment, and Hit@k are the most commonly used evaluation methods.

Compounds
Compounds refer to the small molecules with which the A549 cells (lung cancer) cells are perturbed. In this context, perturbations can be used synonymously to compounds since the cells are perturbed with different compounds. The unique Broad sample ID identifies them (e.g., "BRD-K70301876-034-13-7").

Classes
The weakly supervised classification problem, on which the neural nets are trained, needs to attribute a cell to a specific class. Each class is a different compound in this case. In other experiments, a class could also refer to a genetic perturbation or expression. However, this thesis will always refer to compounds.

MOA
The term *mechanism of action* or *method of action* refers to the specific biochemical interaction through which a drug produces its pharmacological effect. In the LINCS database, compounds are labeled with MOAs, allowing us to group compounds by their biochemical effect.

MOA class size
The MOA class size is the number of compounds that are labeled with a given MOA. In the LINCS dataset, some MOAs only occur once (MOA class size 1), and some MOAs can be found on 20 different compounds (MOA class size 20). This inequality is relevant for discussions on evaluation metrics.

Technical replicates, same-compound wells, and same-MOA compounds
A technical replicate of a well is an exact replicate with the same perturbation and dosage. All negative controls are thus technical replicates of each other. Given a well X, the same-compound well Y is a well that was perturbed with the same compound but a different dosage. Finally, given compound X and its profile, a same-MOA compound is another compound Y that shares the same MOA. The exact wording may depend on context, but I will refer to these same-MOA compounds as MOA matches or MOA pairs.

Profiles
Profiles refer to a set of features that describe the state and morphology of a cell or a group of cells. The term stems from the biological method (profiling) from which these features are extracted. In this thesis, profiles are categorized by their level (see below) and can be the output of CellProfiler and DeepProfiler.

Target
In the drug-discovery process, a target is a receptor, enzyme, or protein therapeutically relevant in curing and treating a disease or dysfunction.

Plates
The LINCS data was acquired by setting cells and perturbations onto tray-like objects (plates) with 384 small test tubes (wells).

Well
A well is a small compartment within a plate that holds the cells and their perturbation material. If two wells have identical contents, this is referred to as a technical replica.

Sites
When imaging the results of a high-throughput plate experiment, the microscopes may take several images of a well to cover the full area of cells. The LINCS dataset has nine sites for each well.

Platemap
Platemaps describe the arrangement of compounds on a given plate. This metadata information can be found on the LINCS repository.

Profile Levels (1-5)
The following image exemplifies the different profile levels. Level 1 contains the original five-channel

images from each site. Level 2 data refers to the profiles (feature vectors) of each cell, i.e., single-cell profiles. At Level 3, all cells in a well have been aggregated to one profile, i.e., the well level. At this level, the metadata is also added to each profile. Next, at Level 4, the data is normalized (Level 4a) and a feature selection process is employed (4b). Finally, well profiles are aggregated once more to the consensus level, where each profile represents one compound with a specific dosage (Level 5, consensus level).

CHTC

The Center for High Throughput Computing (CHTC) is the computing service department of the University of Wisconsin-Madison. It has given me access to the newest graphic cards to perform my experiments.

Appendices

A. Supplementary experiments

Polypharmacology

If a compound has multiple MOA annotations, these are concatenated and become its own unique MOA. Thus a compound with *MOA_a & MOA_b* does not MOA match with a compound with (only) *MOA_a*. The number of compounds tagged with these "composite" MOAs are listed in Table A.1.

Polycount	n
1	1304
2	170
3	31
4	8
5	1
6	4

Table A.1.: Distribution of Polypharmacology in the LINCS dataset. 1304 compounds have only one MOA, 170 have two, etc.

Example cell images

Figure A.1 shows a single cell through the five channels of Cell Painting and Figure A.2 presents two different sites of the AGP (actin, golgi, plasma membrane) channel. The final Figure A.3 displays the cropped and concatenated single-cell images created by the DeepProfiler export function.

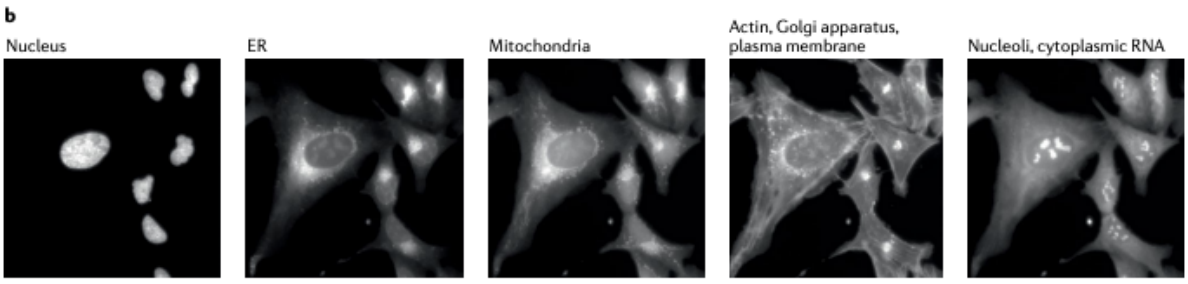

Figure A.1.: *Cell Painting assay images.* Example images from the Cell Painting with its five channels [6]. ER, endoplasmic reticulum.

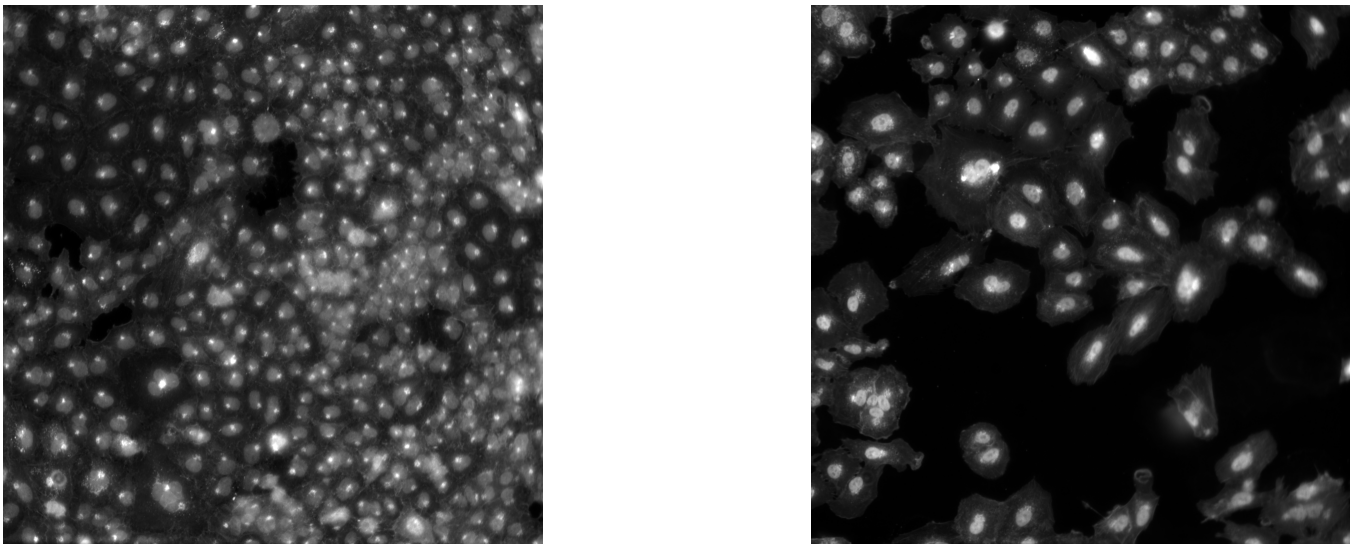

Figure A.2.: *Cell Painting assay sites.* Example site of the AGP Cell Painting channel. Left: healthy, unperturbed cells, in large numbers. Right: perturbed cells that have been diminished in number

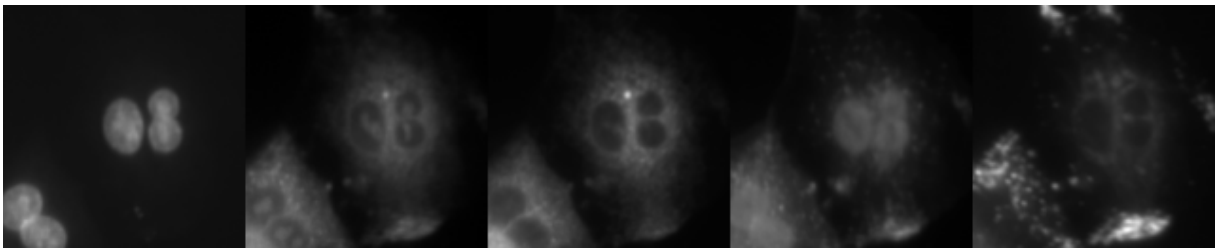

Figure A.3.: *DeepProfiler cropped images.* The DeepProfiler export function outputs concatenated cropped single cells (same cell, each image from one channel). These crops are later randomly accessed during training.

Random initialization

For all other experiments, I have always chosen to initialize the model weights with the pre-trained weights from ImageNet. However, in this experiment I started from a random initialization. The difference in performance between the two initialization schemes is quite large. In general, pre-trained weights outperform random initializations and also achieve higher training and validation accuracy.

	Odds ratio at 99.5%	Precision@R
Random initialization	10.117	0.0481
Pretrained initialization	15.118	0.0615

Table A.2.: Performance of a randomly initializatized model.

Precision@R

Figure A.4 and many other experiments provide evidence that Precision@R agrees on the best performing profiles with *Precision@K* and *Recall@K* in almost all cases. Given this, I found it sufficient to only report *Precision@R*, in addition to *Enrichment* at the 99.5% percentile

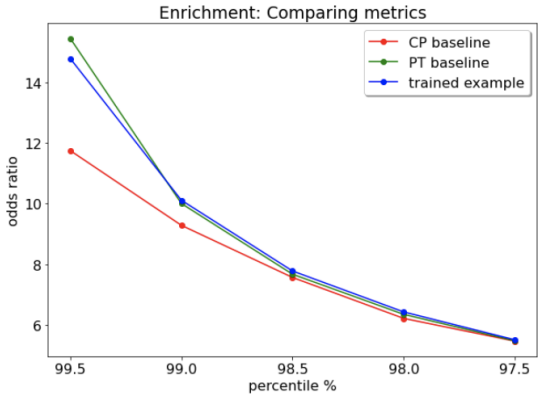

Figure A.4.: *Comparing Precision@K with Precision@R.*

Precision@R values:
PT baseline: 0.072
CP Mad robustize: 0.067
CP Spherized: 0.056

Additional PCA plots

The following PCA plot stems from a randomly initialized model. Visually, it shows no differences from those initialized with EfficientNet.

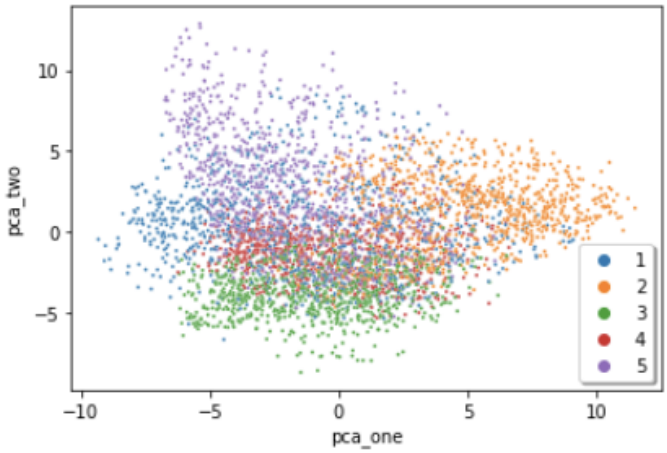

Figure A.5.: *PCA plot of a random initialized model.* Each point represents a well profile and each color denotes a batch.

High learning rates

Very high learning rates lead to un-smooth convergence behavior. In Figure A.6, the learning rate of all three training runs was set to cosine 0.04. Thus, at epoch five, the learning rate was 0.04. This creates a short moment of non-convergence until the learning rate decreases again in the next epoch. In the later stages of the project, I improved the learning schedule to be a step function of slowly reducing learning rates.

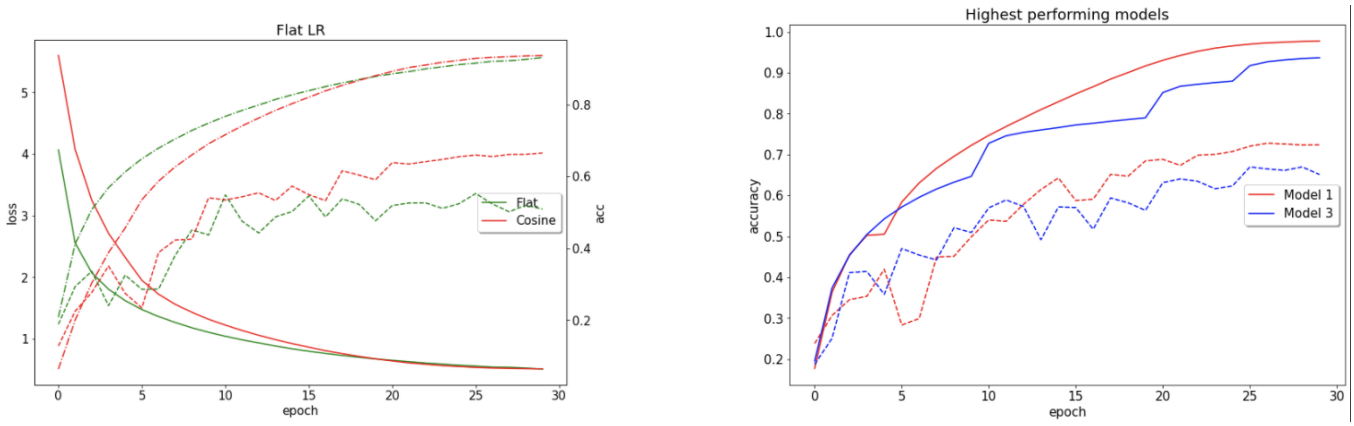

Figure A.6.: *High learning rates.* Left: High constant learning rates (flat curve) lead to unstable convergence of the validation accuracy. Experiment IDs: flat, ID 1022; cosine, ID 1008. Right: Model 1 with a high cosine learning rate displays a 'bump' at the fifth epoch. Model 3 is a custom learning schedule with 4 different learning rates. Experiment IDs: Model 1, ID 1021; Model 3, ID 1103.

B. List of Experiments with hyperparameter details

Some profiles are used throughout the Experiment chapter as baselines. They will be labeled with distinctive names such as CP baseline and pre-trained baseline. Other profiles only occur in a certain section of the Experiment chapter. I refer to these in the order in which they appear and with the exact label as found in the figure. I will add the permanent Github link pointing to the relevant Level 3 or Level 5 CSV file allowing others to directly compare their results to my data. For all trained profiles, the downstream pipeline was spherizing, then median consensus operation.

Hyperparameters

When plotting profiles from trained models, I will report the hyperparameters of each training experiment. The following parameters will be seen as base parameters and only reported if different from the base.

- Architecture = EfficientNet
- Initialization = ImageNet
- Augmentation = False
- Learning rate schedule: cosine
- Label smoothing = 0

The other parameters will always be reported, as they change for most experiments:

- Epochs
- Batch size
- Learning rate
- Training data: see next section

Training data

Here I describe the size and content of the training data (TD). Total single-cells is the number of total cells in the training set. Cells per Epoch is the average number of cells used in one training epoch.

TD1: Top50 MOAs, 512 compounds
Train: cells from three batches
Test: cells from 2 sites from one batch
Total single-cells: 7,575,644
Cells per Epoch: 3,386,880
raw data

TD2: all compounds
Train: cells from three batches
Test: cells from 2 sites from one batch
Total single-cells: 10,544,180
Cells per Epoch: 6,679,098
raw data

TD3: in-SP subset compounds
Train: cells from three batches
Test: cells from 2 sites from one batch
Total single-cells: 6,358,671
Cells per Epoch: 2,624,400
raw data

TD4: ∼450 randomly selected compounds
Train: random cells from all batches
Test: random cells from all batches
Total single-cells: 456,533
Cells per Epoch: 456,533
raw data

TD4.1: ∼450 randomly selected compounds
Train: random cells from all batches
Test: random cells from all batches
Total single-cells: 911,042
Cells per Epoch: 911,042
raw data

TD4.2: ∼450 randomly selected compounds
Train: random cells from all batches
Test: random cells from all batches
Total single-cells: 2,267,640
Cells per Epoch: 2,267,640
raw data

TD5: all compounds
Train: random cells from all batches
Test: random cells from all batches
Total single-cells: 534,166
Cells per Epoch: 534,166
raw data

List of Experiments

In order of appearance.

CP baseline | ID: 100

CellProfiler profile from the LINCS repository - See details. Raw data
Downstream Pipeline: Mad robustized by DMSO, feature selected, spherized, median consensus operation

Pre-trained baseline (PT baseline) | ID: 200

Profiles from the ImageNet pre-trained EfficientNetB0 model. Raw data
Downstream Pipeline: spherized, then median consensus operation

Trained example | ID 813

This is a relatively high-performing example trained model. ID: 813. Raw data
Training parameters: Epochs = 24, Batch size = 64, Learning rate = 0.02, Training data = TD1.

Mad robustize baseline | ID: 101

CellProfiler profile from the LINCS repository - See details. Raw data
Downstream Pipeline: Mad robustized by DMSO, feature selected, spherized, median consensus operation

Trained example | ID 1008

Representative training profiles for pipeline comparison. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.02, Training data = TD4.

ResNet | ID 201

Profiles from the ImageNet pre-trained ResNet model. Raw data
Downstream Pipeline: feature selected, spherized, then median consensus operation

Trained model | ID 913

Model to investigate flat learning schedule. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.02, Training data = TD1, learning rate schedule = constant.

Trained model | ID 910

Model to investigate cosine learning schedule. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.02, Training data = TD1.

Trained model | ID 927

Model to investigate learning rates. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.002, Training data = TD1.

Trained model | ID 917

Model to investigate learning rates. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.005, Training data = TD1.

Trained model | ID 929

Model to investigate learning rates. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.02, Training data = TD1.

Trained model | ID 918

Model to investigate batch sizes. Raw data
Training parameters: Epochs = 20, Batch size = 64, Learning rate = 0.02, Training data = TD1.

Trained model | ID 925

Model to investigate batch sizes. Raw data
Training parameters: Epochs = 20, Batch size = 32, Learning rate = 0.02, Training data = TD1.

Trained model | ID 827

Model to investigate epoch impact on performance. Raw data
Training parameters: Epochs = 8/25, Batch size = 256, Learning rate = 0.02, Training data = TD1.

Trained model | ID 930

Model to investigate epoch impact on performance. Raw data
Training parameters: Epochs = 12, Batch size = 256, Learning rate = 0.01, Training data = TD1, learning rate schedule. = flat.

Trained model | ID 931

Model to investigate epoch impact on performance. Raw data
Training parameters: Epochs = 12-20, Batch size = 256, Learning rate = 0.002, Training data = TD1, learning rate schedule. = flat.

Trained model | ID 1016

Model to investigate label smoothing. Raw data
Training parameters: Epochs = 12-20, Batch size = 256, Learning rate = 0.002, Training data = TD4, label smoothing = 0.05.

Trained model | ID 1009

Model to investigate label smoothing. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.02, Training data = TD4, label smoothing = 0.2.

Trained model | ID 1012

Model to investigate label smoothing. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.02, Training data = TD4, label

smoothing = 0.1.

Trained model | ID 1010

Model to investigate augmentation. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.02, Training data = TD4, augmentation = true.

Trained model | ID 1103

Model to investigate augmentation and label smoothing. Raw data
Training parameters: Epochs = 30, Batch size = 32, Learning rate = 0.02, Training data = TD5, augmentation = off, learning rate schedule = (epoch:[10,20,25], lr:[0.01, 0.005, 0.002]), .

Trained model | ID 1105

Model to investigate augmentation and label smoothing. Raw data
Training parameters: Epochs = 30, Batch size = 64, Learning rate = 0.04, Training data = TD5, augmentation = true.

Trained model | ID 926

Model to investigate SP subset. Raw data
Training parameters: Epochs = 20, Batch size = 64, Learning rate = 0.02, Training data = TD3.

Trained model | ID 921

Model to investigate training on all data. Raw data
Training parameters: Epochs = 25, Batch size = 64, Learning rate = 0.02, Training data = TD2.

Trained model | ID 1021

Model to investigate training on more compounds. Raw data
Training parameters: Epochs = 30, Batch size = 64, Learning rate = 0.02, Training data = TD4.

Trained model | ID 1024

Model to investigate training on more compounds. Raw data
Training parameters: Epochs = 30, Batch size = 64, Learning rate = 0.02, Training data = TD5.

Trained model | ID 1019

Model to investigate training on larger sets. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.02, Training data = TD6.

Trained model | ID 1020

Model to investigate training on larger sets. Raw data
Training parameters: Epochs = 30, Batch size = 256, Learning rate = 0.02, Training data = TD7.

Trained model | ID 1028

Model to investigate batch enrichment scores. Raw data
Training parameters: Epochs = 30, Batch size = 64, Learning rate = 0.04, Training data = TD5, augmentation = true, label smoothing = 0.1.

Trained model | ID 1101

Model to investigate batch enrichment scores. Raw data
Training parameters: Epochs = 30, Batch size = 32, Learning rate = 0.02, Training data = TD5.

List of Figures

List of Tables

Bibliography

1. Bray, M.-A., Singh, S., Han, H., Davis, C. T., Borgeson, B., Hartland, C., Kost-Alimova, M., Gustafsdottir, S. M., Gibson, C. C. & Carpenter, A. E. Cell Painting, a high-content image-based assay for morphological profiling using multiplexed fluorescent dyes. en. *Nat. Protoc.* **11,** 1757–1774 (Sept. 2016).
2. Carpenter, A. E., Jones, T. R., Lamprecht, M. R., Clarke, C., Kang, I. H., Friman, O., Guertin, D. A., Chang, J. H., Lindquist, R. A., Moffat, J., Golland, P. & Sabatini, D. M. CellProfiler: image analysis software for identifying and quantifying cell phenotypes. en. *Genome Biol.* **7,** R100 (Oct. 2006).
3. DiMasi, J. A., Grabowski, H. G. & Hansen, R. W. Innovation in the pharmaceutical industry: New estimates of R&D costs. en. *J. Health Econ.* **47,** 20–33 (May 2016).
4. Chandrasekaran, S. N., Ceulemans, H., Boyd, J. D. & Carpenter, A. E. Image-based profiling for drug discovery: due for a machine-learning upgrade? en. *Nat. Rev. Drug Discov.* (Dec. 2020).
5. Pratapa, A., Doron, M. & Caicedo, J. C. Image-based cell phenotyping with deep learning. en. *Curr. Opin. Chem. Biol.* **65,** 9–17 (May 2021).
6. Gustafsdottir, S. M., Ljosa, V., Sokolnicki, K. L., Anthony Wilson, J., Walpita, D., Kemp, M. M., Petri Seiler, K., Carrel, H. A., Golub, T. R., Schreiber, S. L., Clemons, P. A., Carpenter, A. E. & Shamji, A. F. Multiplex cytological profiling assay to measure diverse cellular states. en. *PLoS One* **8,** e80999 (Dec. 2013).
7. Caicedo, J. C., Cooper, S., Heigwer, F., Warchal, S., Qiu, P., Molnar, C., Vasilevich, A. S., Barry, J. D., Bansal, H. S., Kraus, O., Wawer, M., Paavolainen, L., Herrmann, M. D., Rohban, M., Hung, J., Hennig, H., Concannon, J., Smith, I., Clemons, P. A., Singh, S., Rees, P., Horvath, P., Linington, R. G. & Carpenter, A. E. Data-analysis strategies for image-based cell profiling. en. *Nat. Methods* **14,** 849–863 (Aug. 2017).
8. Huang, K. & Murphy, R. F. *Automated classification of subcellular patterns in multicell images without segmentation into single cells* in *2004 2nd IEEE International Symposium on Biomedical Imaging: Nano to Macro (IEEE Cat No. 04EX821)* (Apr. 2004), 1139–1142 Vol. 2.
9. Chebira, A., Barbotin, Y., Jackson, C., Merryman, T., Srinivasa, G., Murphy, R. F. & Kovacević, J. A multiresolution approach to automated classification of protein subcellular location images. en. *BMC Bioinformatics* **8,** 210 (June 2007).
10. Fuchs, F., Pau, G., Kranz, D., Sklyar, O., Budjan, C., Steinbrink, S., Horn, T., Pedal, A., Huber, W. & Boutros, M. Clustering phenotype populations by genome-wide RNAi and multiparametric imaging. en. *Mol. Syst. Biol.* **6,** 370 (June 2010).
11. Sommer, C. & Gerlich, D. W. Machine learning in cell biology - teaching computers to recognize phenotypes. en. *J. Cell Sci.* **126,** 5529–5539 (Dec. 2013).
12. Pawlowski, N., Caicedo, J. C., Singh, S., Carpenter, A. E. & Storkey, A. *Automating Morphological Profiling with Generic Deep Convolutional Networks* en. Nov. 2016.

13. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. *ImageNet: A large-scale hierarchical image database* in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (June 2009), 248–255.
14. Krizhevsky, A., Sutskever, I. & Hinton, G. E. *ImageNet Classification with Deep Convolutional Neural Networks* in *Advances in Neural Information Processing Systems* (eds Pereira, F., Burges, C. J. C., Bottou, L. & Weinberger, K. Q.) **25** (Curran Associates, Inc., 2012).
15. Tan, M. & Le, Q. *Efficientnet: Rethinking model scaling for convolutional neural networks* in *International Conference on Machine Learning* (2019), 6105–6114.
16. Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M. & Thrun, S. Dermatologist-level classification of skin cancer with deep neural networks. en. *Nature* **542,** 115–118 (Feb. 2017).
17. Moen, E., Bannon, D., Kudo, T., Graf, W., Covert, M. & Van Valen, D. Deep learning for cellular image analysis. en. *Nat. Methods* **16,** 1233–1246 (Dec. 2019).
18. Mullard, A. Machine learning brings cell imaging promises into focus. en. *Nat. Rev. Drug Discov.* **18,** 653–655 (Aug. 2019).
19. Zhong, G., Wang, L.-N., Ling, X. & Dong, J. An overview on data representation learning: From traditional feature learning to recent deep learning. *The Journal of Finance and Data Science* **2,** 265–278 (Dec. 2016).
20. Qian, W. W., Xia, C., Venugopalan, S., Narayanaswamy, A., Peng, J. & Michael Ando, D. *Batch Equalization with a Generative Adversarial Network* en. Feb. 2020.
21. Zhang, H., Cisse, M., Dauphin, Y. N. & Lopez-Paz, D. mixup: Beyond Empirical Risk Minimization. arXiv: 1710.09412 [cs.LG] (Oct. 2017).
22. Jaiswal, A., Babu, A. R., Zadeh, M. Z., Banerjee, D. & Makedon, F. *A Survey on Contrastive Self-Supervised Learning* 2020.
23. Han, W., Cheng, Y., Chen, J., Zhong, H., Hu, Z., Chen, S., Zong, L., King, I., Gao, X. & Li, Y. *Self-supervised contrastive learning for integrative single cell RNA-seq data analysis* en. July 2021.
24. Zbontar, J., Jing, L., Misra, I., LeCun, Y. & Deny, S. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. arXiv: 2103.03230 [cs.CV] (Mar. 2021).
25. Becker, T., Yang, K., Caicedo, J. C., Wagner, B. K., Dancik, V., Clemons, P., Singh, S. & Carpenter, A. E. *Predicting compound activity from phenotypic profiles and chemical structures* en. Dec. 2020.
26. Godec, P., Pančur, M., Ilenič, N., Čopar, A., Stražar, M., Erjavec, A., Pretnar, A., Demšar, J., Starič, A., Toplak, M., Žagar, L., Hartman, J., Wang, H., Bellazzi, R., Petrovič, U., Garagna, S., Zuccotti, M., Park, D., Shaulsky, G. & Zupan, B. Democratized image analytics by visual programming through integration of deep models and small-scale machine learning. en. *Nat. Commun.* **10,** 4551 (Oct. 2019).
27. Goldsborough, P., Pawlowski, N., Caicedo, J. C., Singh, S. & Carpenter, A. E. *CytoGAN: Generative Modeling of Cell Images* en. Dec. 2017.
28. Doan, M., Sebastian, J. A., Caicedo, J. C., Siegert, S., Roch, A., Turner, T. R., Mykhailova, O., Pinto, R. N., McQuin, C., Goodman, A., Parsons, M. J., Wolkenhauer, O., Hennig, H., Singh, S., Wilson, A., Acker, J. P., Rees, P., Kolios, M. C. & Carpenter, A. E. Objective assessment of stored blood quality by deep learning. en. *Proc. Natl. Acad. Sci. U. S. A.* **117,** 21381–21390 (Sept. 2020).
29. Caicedo, J. C., McQuin, C., Goodman, A., Singh, S. & Carpenter, A. E. Weakly Supervised Learning of Single-Cell Feature Embeddings. en. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* **2018,** 9309–9318 (June 2018).

30. Fischer, B., Sandmann, T., Horn, T., Billmann, M., Chaudhary, V., Huber, W. & Boutros, M. A map of directional genetic interactions in a metazoan cell. en. *Elife* **4** (Mar. 2015).

31. Kuhn, M. A Short Introduction to the caret Package. *R Found Stat Comput* (2015).

32. Malo, N., Hanley, J. A., Cerquozzi, S., Pelletier, J. & Nadon, R. Statistical practice in high-throughput screening data analysis. en. *Nat. Biotechnol.* **24,** 167–175 (Feb. 2006).

33. Van de Wiel, M. A., Brosens, R., Eilers, P. H. C., Kumps, C., Meijer, G. A., Menten, B., Sistermans, E., Speleman, F., Timmerman, M. E. & Ylstra, B. Smoothing waves in array CGH tumor profiles. en. *Bioinformatics* **25,** 1099–1104 (May 2009).

34. Simm, J., Klambauer, G., Arany, A., Steijaert, M., Wegner, J. K., Gustin, E., Chupakhin, V., Chong, Y. T., Vialard, J., Buijnsters, P., Velter, I., Vapirev, A., Singh, S., Carpenter, A. E., Wuyts, R., Hochreiter, S., Moreau, Y. & Ceulemans, H. Repurposing High-Throughput Image Assays Enables Biological Activity Prediction for Drug Discovery. en. *Cell Chem Biol* **25,** 611–618.e3 (May 2018).

35. Rohban, M. H., Fuller, A. M., Tan, C., Goldstein, J. T., *et al.* Discovery of small molecule pathway regulators by image profile matching. *bioRxiv* (2021).

36. Perlman, Z. E., Slack, M. D., Feng, Y., Mitchison, T. J., Wu, L. F. & Altschuler, S. J. Multidimensional drug profiling by automated microscopy. en. *Science* **306,** 1194–1198 (Nov. 2004).

37. Mechanism matters. en. *Nat. Med.* **16,** 347 (Apr. 2010).

38. Loo, L.-H., Wu, L. F. & Altschuler, S. J. *Image-based multivariate profiling of drug responses from single cells* 2007.

39. Zhu, Y., Yeung, T.-L., Sheng, J., Hinchcliff, E. M., Burks, J. K., Jazaeri, A. A., Mok, S. C. & Wong, S. T. C. *An image informatics pipeline for imaging mass cytometry to characterize the immune landscape in pre- and on-treatment immune therapy and its application in recurrent platinium-resistant epithelial ovarian cancer* 2019.

40. Chen, H., Engkvist, O., Wang, Y., Olivecrona, M. & Blaschke, T. The rise of deep learning in drug discovery. *Drug Discov. Today* **23,** 1241–1250 (June 2018).

41. Natoli, T., Way, G., Lu, X., Cimini, B., Logan, D., Karhohs, K., Caicedo, J., Alimova, M., Hartland, K., Golub, T., Carpenter, A., Singh, S. & Subramanian, A. *broadinstitute/lincs-cell-painting: Full release of LINCS Cell Painting dataset* June 2021.

42. Way, G. P., Natoli, T., Adeboye, A., Litichevskiy, L., Yang, A. X., *et al.* Morphology and gene expression profiling provide complementary information for mapping cell state. *bioRxiv* (2021).

43. Haghighi, M., Singh, S., Caicedo, J. & Carpenter, A. *High-Dimensional Gene Expression and Morphology Profiles of Cells across 28,000 Genetic and Chemical Perturbations* en. Sept. 2021.

44. Stirling, D. R., Swain-Bowden, M. J., Lucas, A. M., Carpenter, A. E., Cimini, B. A. & Goodman, A. CellProfiler 4: improvements in speed, utility and usability. en. *BMC Bioinformatics* **22,** 433 (Sept. 2021).

45. Müller, R., Kornblith, S. & Hinton, G. When Does Label Smoothing Help? arXiv: 1906.02629 [cs.LG] (June 2019).

46. Ando, M. D., McLean, C. & Berndl, M. *Improving Phenotypic Measurements in High-Content Imaging Screens* en. July 2017.

47. Kessy, A., Lewin, A. & Strimmer, K. Optimal Whitening and Decorrelation. *Am. Stat.* **72,** 309–314 (Oct. 2018).

48. Schütze, H., Manning, C. D. & Raghavan, P. *Introduction to information retrieval* (Cambridge University Press Cambridge, 2008).

49. Zhu mu (2004). recall precision and average ... - For Rent Leuven.

50. Rohban, M. H., Abbasi, H. S., Singh, S. & Carpenter, A. E. Capturing single-cell heterogeneity via data fusion improves image-based profiling. en. *Nat. Commun.* **10,** 2082 (May 2019).

51. German, Y., Vulliard, L., Kamnev, A., Pfajfer, L., Huemer, J., Mautner, A.-K., Rubio, A., Kalinichenko, A., Boztug, K., Ferrand, A., Menche, J. & Dupré, L. Morphological profiling of human T and NK lymphocytes by high-content cell imaging. en. *Cell Rep.* **36,** 109318 (July 2021).