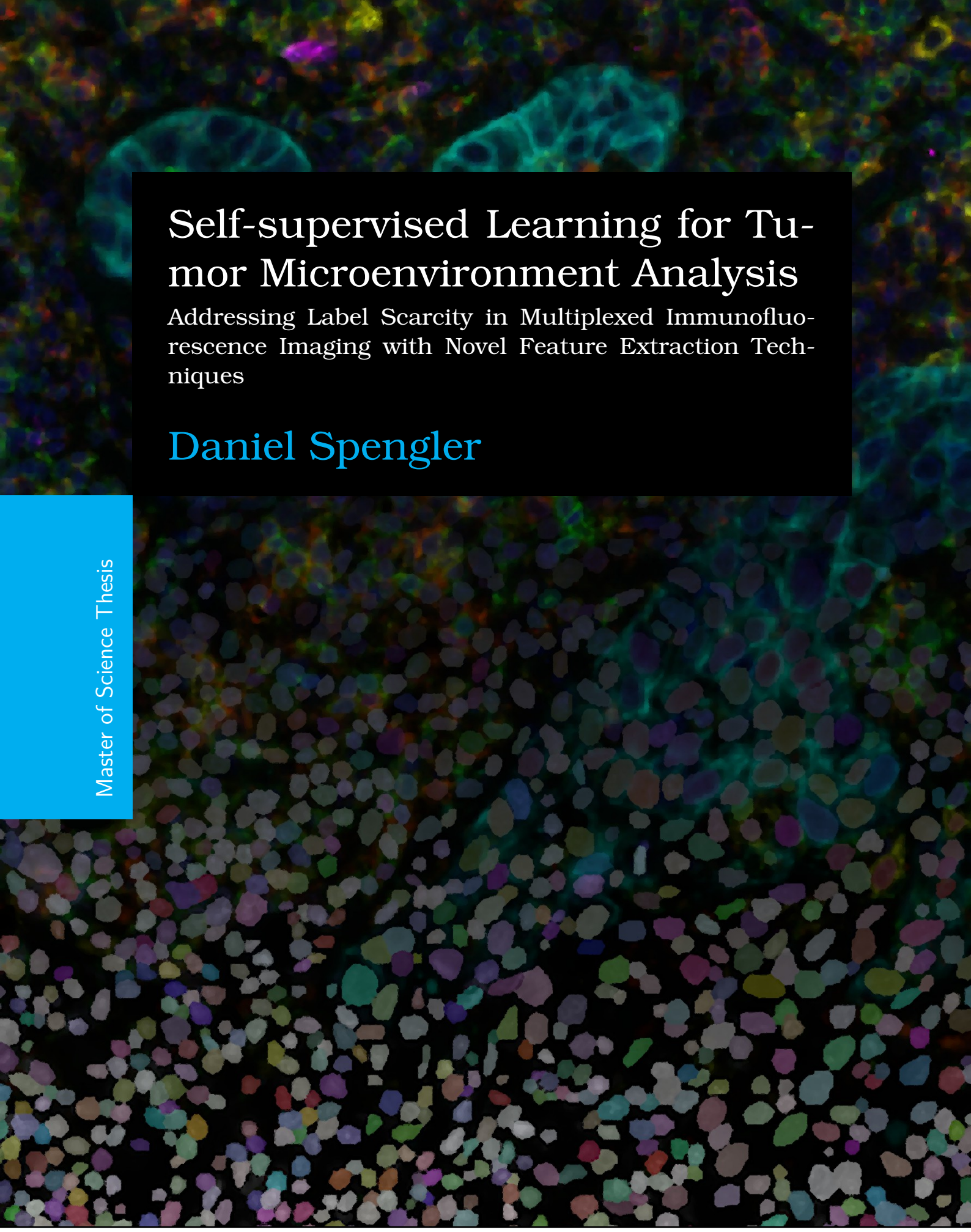


Self-supervised Learning for Tumor Microenvironment Analysis

Addressing Label Scarcity in Multiplexed Immunofluorescence Imaging with Novel Feature Extraction Techniques

Daniel Spengler

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Self-supervised Learning for Tumor Microenvironment Analysis

Addressing Label Scarcity in Multiplexed Immunofluorescence Imaging with Novel Feature Extraction Techniques

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Daniel Spengler

May 8, 2023

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis is part of a joint collaboration between Delft University of Technology (TU Delft) and Erasmus University Medical Center (Erasmus MC). Furthermore, the work was supported by Erasmus MC's Academic Center of Excellence for Tumor Immunology and Immune Therapy (ACE TI-IT). Their support is hereby gratefully acknowledged.

Code Availability Statement
The code used in this thesis is not currently available for public use.

Abstract

The study of tumor microenvironments (TMEs) and immune cell composition in cancer, a disease characterized by uncontrolled growth and spread of tumor cells, has become increasingly important for understanding tumor progression and patient outcomes. Tools such as the TME-Analyzer enable this kind of research, but their manual workflows highlight a common problem in medical imaging: the scarcity of labeled data. This limits the efficiency and applicability of supervised learning algorithms to improve such medical image analysis tools. Self-supervised learning algorithms offer a promising alternative by learning feature representations without requiring labeled data. This thesis aims to address the issue of label scarcity by exploring the potential of self-supervised learning models for TME analysis involving the classification of individual cells in multiplex immunofluorescence (MxIF) microscopy images of triple-negative breast cancer (TNBC) tissue.

To enable the learning of feature representations from MxIF images with an arbitrary number of color channels, this thesis proposes to pre-train an encoder network on every image channel separately according to the SimCLR algorithm and perform classification of multi-channel images by feeding the concatenated feature representation outputs of every channel to a classifier network — referred to as the Siamese configuration. A hyperparameter search is conducted to optimize the SimCLR encoder's ability to learn high-quality feature representations of individual cells in MxIF images of TNBC tissue. Upon obtaining an optimal set of hyperparameters, the effectiveness of the learned feature representations in improving label-efficiency for individual cell classification is assessed.

The results demonstrate that the proposed Siamese configuration improves the accuracy of classifying the inflammation status of TNBC tumor sections by 2.63 %. Additionally, the optimal set of hyperparameters identified through the search include the use of the normalized temperature cross-entropy loss function with low temperature and an added image intensity thresholding term, as well as zoom and brightness/contrast augmentations. Furthermore, the optimized self-supervised learning model improves label-efficiency for individual cell classification, maintaining performance with only 40 % of labeled data, while performance drops only when the label percentage is reduced below this threshold.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cancer is a class of diseases characterized by uncontrolled growth and spread of tumor cells throughout the body [1]. It is one of the leading causes of death worldwide and research into its prevention, detection, and treatment remains a challenge in the medical field to this day. The most widely used system for classifying cancers is the tumor-node-metastasis (TNM) staging system, which categorizes cancers by their tumor size, lymph nodes affected, and severity of metastasis [2]. It is one of the factors taken into account by oncologists for cancer prognosis, i.e., the estimate of how the cancer disease will proceed, and decisions about treatment.

Besides the aforementioned markers of the TNM staging system, the composition of the tumor microenvironment (TME), including the presence of certain immune cells, has been shown to be a predictor of clinical outcomes in various types of cancers, such as colorectal cancer [3] and triple-negative breast cancer (TNBC) [4]. Moreover, these TME-based biomarkers, particularly the immune response — which includes immune cell abundance, signaling mechanisms, and non-immune stromal cells that might affect the immune response — may be superior predictors compared to the TNM, where identically classified cancers across different patients can have very different clinical outcomes [5]. This has led to discussions in the field of oncology and cancer research about appending the TNM staging system with classification systems based on the TME and immune response, such as the immune contexture or Immunoscore [5].

The Immunoscore's accuracy relies on the ability to detect, count, and locate multiple biomarkers in a large amount of samples. The need to identify the presence of certain immune cells in specific locations of the TME means that tissue-based imaging methods are more suitable than bulk-assays such as flow cytometry, which destroys the biopsy's spatial architecture. The promising predictive capabilities of such immune cell based classification systems create a need for improved digital pathology technology that can address the requirements for sophisticated image analysis as well as high throughput [6].

A recently developed tool for this purpose is the TME-Analyzer by Balcioglu et al. [7] at Erasmus MC's (Erasmus University Medical Center) TME Facility. The TME-Analyzer provides an interactive graphical user interface that enables the analysis of immune cell populations within tumors and their spatial distribution, which is critical for understanding tumor

evolution and developing new treatments. In a case study of a cohort of TNBC patients [4], the TME-Analyzer was used to identify specific characteristics of the immune cell populations that were related to patient survival.

Motivated by the work performed by the colleagues at Erasmus MC, this thesis is a collaborative effort between Erasmus MC and TU Delft that investigates the application of self-supervised learning models for TME analysis in order to address the issue of label scarcity in medical imaging, with the potential to enhance tools such as the TME-Analyzer.

Thesis Motivation: Addressing Label Scarcity in Medical Imaging — A Self-Supervised Learning Approach

Advanced image analysis algorithms based on Deep Learning (DL) are gaining traction in medical fields [8, 9, 10, 11]. Ideally, such algorithms are trained via supervised learning, i.e. they learn from many annotated examples. This way, an immediate learning signal can be provided during training, which the algorithm can optimize towards. However, a disadvantage of supervised learning is that it heavily relies on the quality and quantity of labeled data available during training. This is particularly critical in the context of medical imaging, since experts are required to perform such labeling, which can be costly and time-consuming. In fact, the time-consuming nature of using the TME-Analyzer, which is essentially a labeling tool for medical images, is one of the key issues we would aim to address by equipping it with DL capabilities.

To overcome the limitations of supervised learning, self-supervised learning has gained popularity[1] as a promising alternative [13]. This approach involves training an algorithm on a pretext task to learn feature representations of the data without requiring any labels. One example of a powerful self-supervised learning algorithm for image processing is the Simple Framework for Contrastive Learning of Visual Representations (SimCLR) [14], which learns feature representations by contrasting augmented versions of the same image. The resulting learned representations of the data can serve as basis for training other models to perform downstream tasks such as classification, detection, and segmentation, and can be more label-efficient than supervised learning approaches. SimCLR has shown promising results in medical image analysis [15] where it was used to improve label-efficiency for tasks such as tumor classification and segmentation [16], individual cell segmentation [17], and the identification of novel TMEs [18].

As a result of their study of validating the TME-Analyzer [7], the colleagues from Erasmus MC have obtained a labeled dataset of multiplex immunofluorescence (MxIF) tissue images from TNBC patients [4]. Rather than using these labels to train a model via supervised learning, I could instead aim to train a SimCLR model that learns useful feature representations from the image data. If successful, this approach could be generalized to learning feature representations from similar images outside the TNBC dataset, making it useful in future applications where no labels are available. Furthermore, the resulting model could then be integrated into the TME-Analyzer, enhancing its capabilities, assisting users throughout the workflow, and accelerating the overall process.

[1] A particularly popular example of a model that utilizes self-supervised learning is GPT (Generative Pre-trained Transformer) [12]: A powerful language model that was pre-trained on hundreds of gigabytes of text data without requiring any labels.

Since the TNBC dataset consists of MxIF images with 8 color channels, certain considerations must be taken into account. This type of image domain is less commonly addressed in the related literature compared to brightfield images with 3 color channels. Consequently, it is less obvious how to choose hyperparameters for a SimCLR model to successfully learn from the TNBC dataset. For example, image augmentations are an important part of the SimCLR algorithm, but it is unclear if the commonly recommended augmentations [14, 19] work well with MxIF images. Furthermore, although images from the TNBC dataset have 8 color channels, other images that we may wish to analyze using the TME-Analyzer could have different amounts of color channels. The model should therefore be flexible in the amount of image channels it accepts as its input.

Research Questions

The previous considerations lead me to the following research questions:

- *How can a self-supervised learning model be designed to learn features from MxIF images with an arbitrary number of color channels?*
- *What are the hyperparameters for such a model that improve feature learning of individual cells in MxIF images?*
- *Can the features learned by such a model improve label-efficiency for the task of individual cell classification?*

To answer the research questions, a systematic approach can be employed:

Since the model's performance cannot be compared against actual clinical outcomes within the scope of this thesis, the evaluation will rely on comparing the results to the labels provided by the TME-Analyzer. If the model approaches the performance of the TME-Analyzer, this would be considered validation of its effectiveness.

In order to assess the model's ability to learn features from MxIF images and the effect of hyperparameters, the change in performance can be tracked as the model's hyperparameters are adjusted and optimized. If performance improves, this would provide answers to the questions concerning the model's ability to learn features from MxIF images and the impact of hyperparameters.

Lastly, to address the question of label-efficiency, the final model's performance can be assessed in relation to the amount of labeled data required. If the model can achieve higher performance with fewer labels compared to a previous iteration, this would provide evidence that the self-supervised learning model improves label-efficiency for the task of individual cell classification.

Training Pipeline Overview

An overview of the proposed training pipeline is shown in Figure 1-1. The reader is advised to return to this figure and study it in more detail after reading the individual chapters of this thesis.

Document Structure

The remainder of this document is structured as follows:

Chapter 2 provides necessary background information: Section 2-1 starts with an overview of microscopy imaging and tissue analysis. Section 2-2 gives an introduction to basic concepts in machine learning, followed by Section 2-3, exploring key components and techniques used in deep learning. Section 2-4 gives a brief overview of self-supervised learning and highlights the Simple Framework for Contrastive Learning of Visual Representations (SimCLR), the self-supervised learning algorithm employed in this work. Lastly, Section 2-5 discusses related work and its limitations.

Chapter 3 outlines the methodology used, detailing the steps taken to design, train, and evaluate the self-supervised learning models. Chapter 4 presents the results obtained from the experiments conducted and Chapter 5 provides a discussion of these results and offers insights into potential future work. Chapter 6 concludes the document by summarizing the main findings and their significance in the context of the research questions.

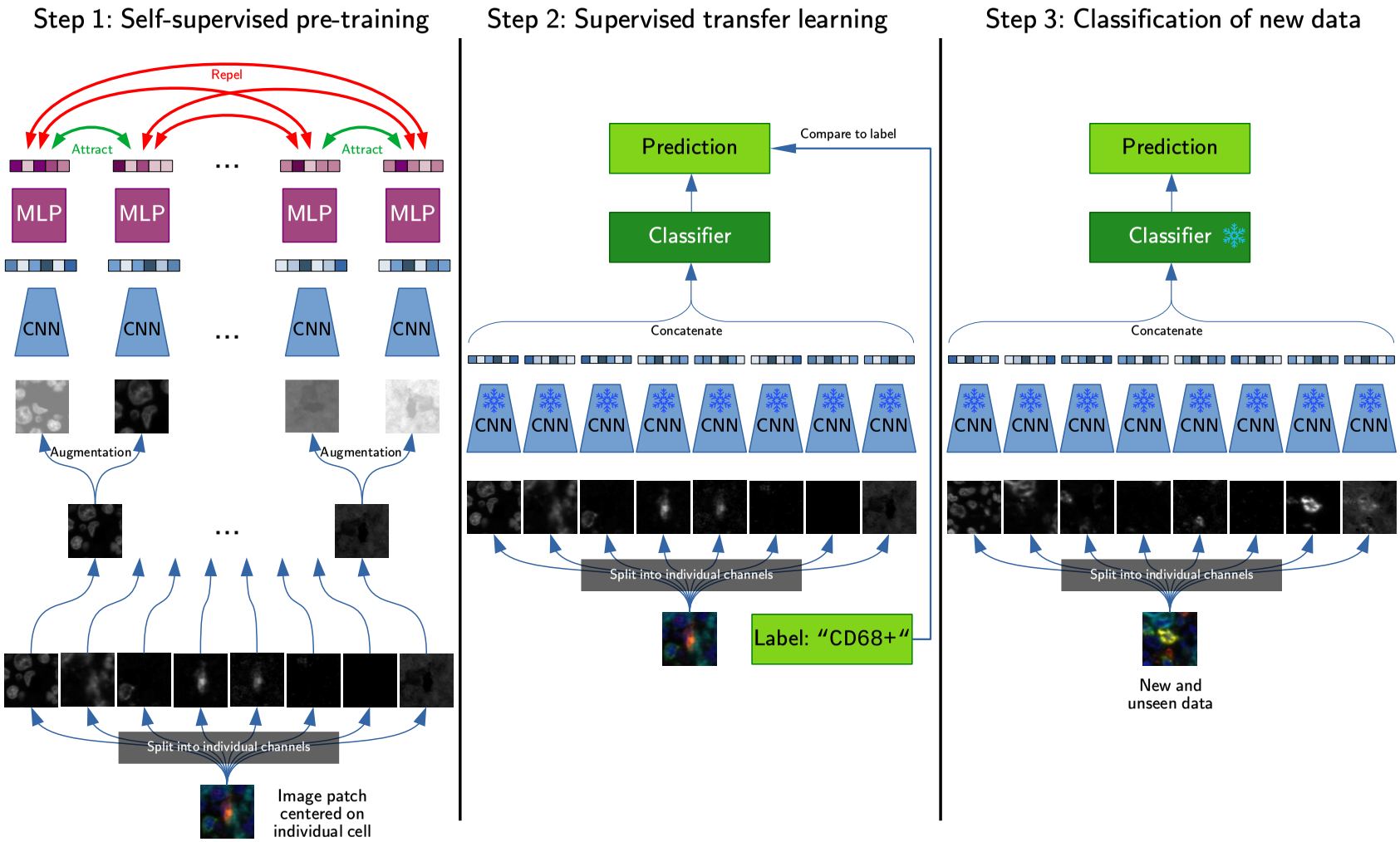


Figure 1-1: Overview of the proposed method to train a SimCLR model for MxIF images with arbitrary number of color channels. In **Step 1**, self-supervised pre-training is performed to obtain an encoder CNN. To this end, input images are split into their separate channels, which get treated as individual images. SimCLR learns feature representations of images by encoding randomly augmented image pairs and minimizing a contrastive loss across all image pairs in the batch. In **Step 2**, the pre-trained encoder CNN's weights are frozen and it is now fed unaugmented images directly. The concatenated feature encodings are passed to a classifier, which is trained to predict the cell's label (supervised learning). In **Step 3**, labels for new and unseen data are predicted. The weights of all networks denoted by MLP and CNN are shared, i.e., **only one** encoder CNN is trained. During pre-training, the amount of channels for each input image can be arbitrary.

Chapter 2

Background Information

This chapter provides the background information needed to understand the methodology and results of this thesis. The content is designed to equip the average MSc student with a solid understanding of the key concepts and techniques involved in this research. The sections cover the following topics:

- Section 2-1 delves into microscopy imaging and tissue analysis, discussing various microscopy techniques and tissue preparation methods, as well as tumor immunology and cellular markers. For a refresher on basic optics concepts, see Appendix B. Detailed specifications of the triple-negative breast cancer (TNBC) dataset used in the experiments can be found in Appendix A.
- Section 2-2 introduces the fundamentals of machine learning, including its paradigms, model evaluation metrics, overfitting, underfitting, generalization, regression, regularization, classification, and the Perceptron. A visual example of the major learning paradigms is available in Appendix E.
- Section 2-3 explores deep learning concepts, such as multi-layer perceptrons, activation functions, backpropagation, and optimization algorithms for weight updates. In particular, this section highlights convolutional neural networks (CNNs) — the deep networks commonly used for computer vision tasks. Appendix C offers a step-by-step derivation of the backpropagation equations, while Appendix D provides a brief review of the state-of-the-art in computer vision over the past decade.
- Section 2-4 presents the intuition behind self-supervised learning and describes the SimCLR (Simple Framework for Contrastive Learning of Visual Representations) algorithm in detail. Furthermore, the concept of linear evaluation — a simple method to evaluate performance of self-supervised models — is explained.
- Section 2-5 reviews related work in the field, including state-of-the-art tools for tumor analysis, applications of SimCLR for tumor analysis, and limitations of previous research.

2-1 Microscopy Imaging and Tissue Analysis

This section will give an overview of basic concepts in microscopy imaging and tissue analysis.

The imaging data that was analyzed as part of this thesis was generated by a fluorescence microscopy imaging system (see Appendix A for a detailed explanation of the dataset) and features images of tumor tissue. As a result, the objects relevant to the analysis are individual cells.

The interested reader may consult Appendix B for a refresher on basic properties of optical systems, but this information is not required to follow the rest of this section.

Section 2-1-1 presents the two microscopy imaging techniques most relevant to this thesis work, namely **bright-field microscopy** and **fluorescence microscopy**.

Section 2-1-2 gives a brief overview of the necessary steps to prepare tissue for imaging, including the most relevant **staining techniques** that are used for bright-field and fluorescence microscopy.

Lastly, Section 2-1-3 elaborates on common terminology used in the context of tumor immunology.

2-1-1 Microscopy Techniques

Various microscopy techniques exist for tissue and cell imaging. Here, we will focus on techniques which make use of tissue staining, namely bright-field microscopy and fluorescence microscopy.

Bright-field Microscopy

Bright-field microscopy is perhaps the most well-known method for microscopic imaging. Figure 2-1 shows the basic configuration: light passing through a condenser lens is illuminating the specimen and continues through an objective lens to be recorded by a sensor or observed through an eyepiece. The specimen to be observed becomes visible by attenuating the light from the source and will appear darker the more opaque it is. Contrast is provided by varying opacity within the specimen, e.g. a cell's walls and its contents, which transmits light in varying intensities. In order to improve contrast, tissue samples may be stained using chromogenic dyes such as hematoxylin and eosin (H&E) before imaging (see Section 2-1-2).

Fluorescence Microscopy

In fluorescence microscopy, specimens are made visible by making use of their fluorescence properties rather than their attenuation of light as is the case with bright-field imaging. A fluorescent substance has the property of absorbing light of a certain wavelength and re-emitting light of a longer wavelength. For example, a fluorescent substance hit by blue light may, in turn, emit green light.

Figure 2-2a shows the basic configuration of a wide-field fluorescence (also known as epi-fluorescence) microscope. Light from the source is filtered in order to illuminate the specimen

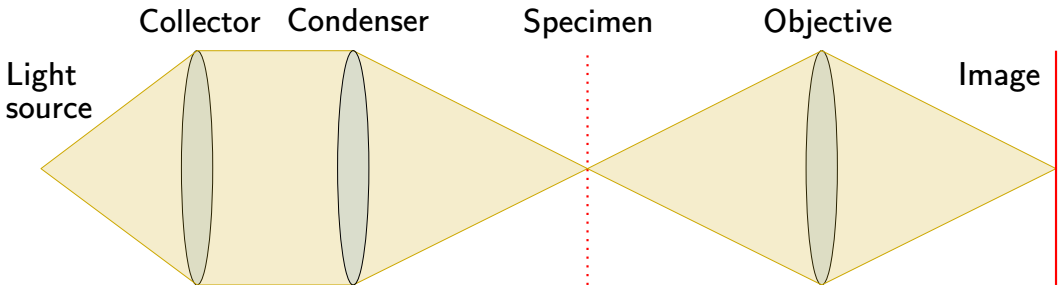


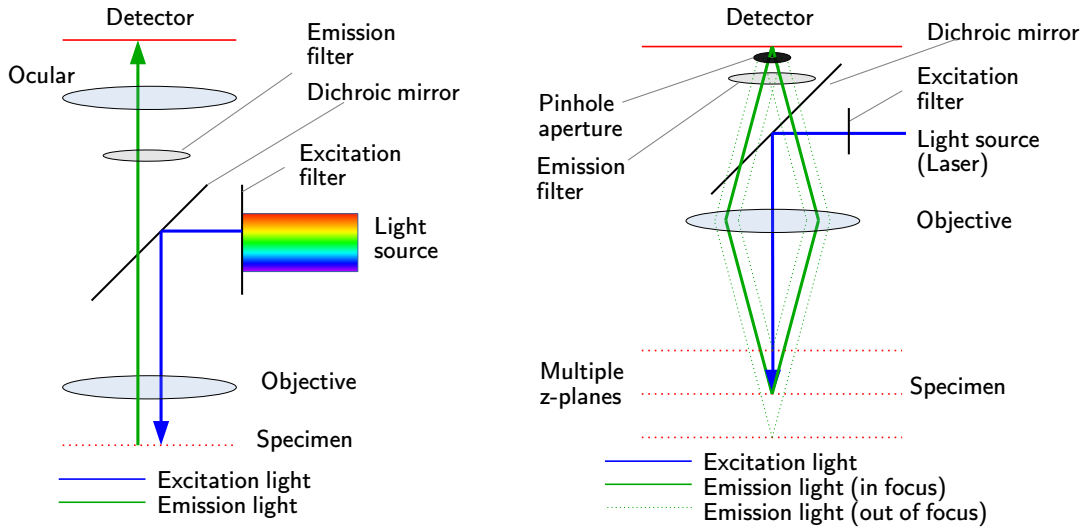
Figure 2-1: Schematic of a bright-field microscope. The light from a light source passes through a collector/condenser configuration and illuminates the specimen from the opposite side of the observer. The objective then focuses the illuminated specimen onto an image. Here, only a basic lens configuration is shown. In practice, configurations similar to the ones shown in Figure B-3 and Figure B-4 are used in order to give more control over focus and magnification.

only with light of a select wavelength. An angled dichroic mirror reflects the light towards the objective where it hits the specimen. The light of higher wavelength (and therefore lower energy) fluoresced by the specimen travels back the same path, but is being transmitted by the dichroic mirror. An additional filter is used to reduce the light to only those wavelengths that are expected to be emitted by the fluorescent specimen and can then be observed through an eyepiece or a digital sensor.

A common variant of the fluorescence microscope is the confocal microscope, whose basic configuration is shown in Figure 2-2b. The light path from the source to the specimen is similar to that in a wide-field fluorescence microscope. However, in confocal fluorescence microscopes, lasers are chosen as light source due to better focusability, allowing for precise spot-illumination of the specimen. The fluoresced light travels back, again being transmitted by the dichroic mirror and filtered by an emission filter, and has to pass through a pinhole located such that only light originating from the desired focal plane reaches the detector.

Wide-field fluorescence microscopes have the advantage of the specimen being immediately visible through an eyepiece, and often allow for simple bright-field illumination as well. Since the entire specimen is exposed to light at once, it is better suited for thin specimens where the expected background fluorescence is low. Confocal fluorescence microscopes eliminate this problem by only illuminating small sections of the specimen at once and only capturing light from the focal plane. However, this means that specimens must first be scanned before a complete image can be produced. By altering the location of the focal plane with respect to the specimen, this scanning procedure can also be extended in the z-direction, allowing for 3D imaging of a specimen.

All fluorescence microscopes require the specimen to be fluorescent. Although biological tissue is autofluorescent to some degree, this source of fluorescence is usually considered noise in practical applications. Instead, tissue samples are treated with fluorescent dyes in order to make them visible to a fluorescent microscope. By utilizing immunohistochemistry (IHC), different dyes can be attached to different types of cells in a process called immunostaining. This is addressed in more detail in Section 2-1-2.



(a) Wide-field fluorescence microscope.

(b) Confocal fluorescence microscope.

Figure 2-2: Schematics of different fluorescence microscope configurations. The configuration of filtering and directing light of certain wavelengths by the use of excitation filters, dichroic mirrors and emission filters is shared by both configurations. A wide-field fluorescence microscope illuminates the entire object plane and a complete image is formed immediately. A confocal fluorescence microscope only illuminates a small point in the object plane and has to scan across the object plane before an image can be generated. The pinhole aperture prevents light emitted from z-planes other than the one in focus to reach the detector.

2-1-2 Tissue Preparation

In order to prepare tissue samples for imaging under a microscope, typically some form of fixation, embedding, and staining are performed [20]. Fixation is the process of thermally and/or chemically treating a sample such that natural biological processes such as decay are halted. The most common chemical agents for this process are aldehydes and alcohols. The result of fixation is that the tissue sample is essentially "killed" while preserving its shape and structure (and possibly even mechanically reinforcing it). Samples are then embedded in wax in order to provide rigidity (an alternative to this is freezing them) for being sliced into thin sections — in the range of 5 µm to 30 µm — such that they can be viewed under a microscope.

Staining is performed in order to improve contrast and visibility of the sample. The type of stain used and the method application depends on which type of microscope is used and which parts of the tissue sample need to be examined in more detail. We can broadly distinguish between conventional staining, which is a relatively straightforward application of a dye or stain to a sample, and immunostaining, which utilizes the relationship between antibodies and antigens in order to target specific cells with a dye or stain.

Conventional Staining

Due to the relatively low contrast of tissue samples, they are stained with chromogenic dyes for the purposes of bright-field microscopy. Although various illumination techniques and microscope configurations to improve contrast exist, staining has the advantage of pronouncing certain cellular features with different dyes, providing not only contrast but also color for better visibility. The most commonly used stain for this purpose is hematoxylin and eosin (H&E) [21]. Hematoxylin colors cell nuclei in shades of purple and blue whereas eosin colors cytoplasm, certain tissue fibers and the extracellular matrix in shades of red and pink. When observed through a microscope, this results in the common purple and pink colored tissue image which many readers may be familiar with (see Figure 2-3a). Due to its widespread use, the majority of pathologists are trained in analyzing H&E stained tissue sections.

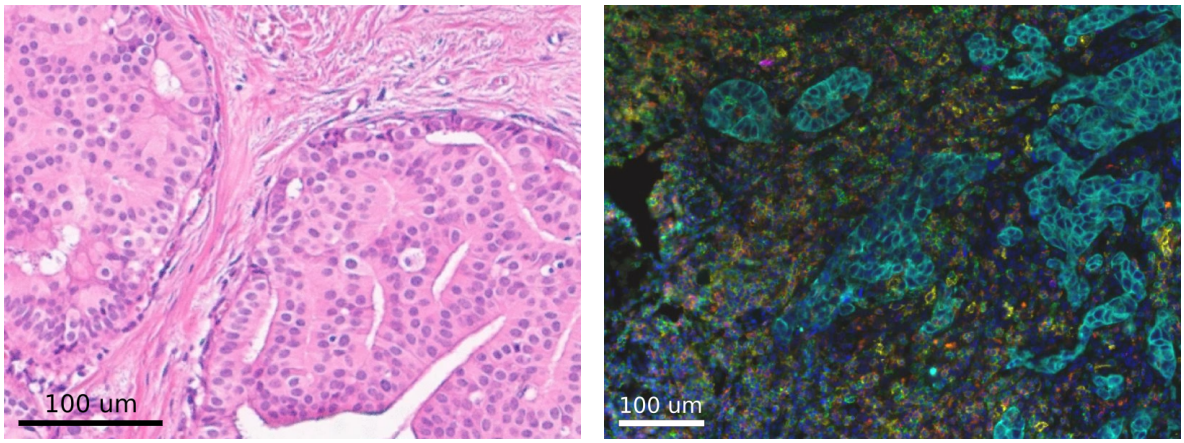
Immunostaining

Another process of staining a sample involves utilizing the immune system's chemistry to attach certain dyes and stains to specific proteins present in tissue. This process is called immunostaining and more specifically, in the case of staining tissue, immunohistochemistry (IHC) [22]. It can be used to stain tissue for bright-field microscopy (chromogenic immunohistochemistry) as well as fluorescence microscopy (immunofluorescence). IHC makes use of the fact that antibodies only attach to a specific antigen. With the knowledge that certain antigens are only presented by certain structures, such as specific cells or the extracellular matrix, corresponding antibodies tagged (or *conjugated*) with a dye, or fluorophores in the case of immunofluorescence, are introduced to the tissue sample and, as a result, make those structures visible to a microscope. This is not only useful to target specific cells, but also to quantify the general amount of cells or extracellular matrix.

By repeating this process using fluorophores with varying light wavelengths for excitation and emission to target different antigens, multiplexed images can be generated, visualizing multiple cell types within the same tissue sample. An example of this can be seen in Figure 2-3b, which shows a 7-plex image of a breast tissue section stained using the Opal workflow [23]. Here, seven individual monochrome image channels are composited such that they can be displayed as a 3-color channel RGB (red green blue) image. This is useful for analysis by humans, but introduces a loss of information from the individual channels. For the purposes of computer vision analysis, it is therefore more useful to analyze the individual monochrome channels individually.

Immunofluorescence staining, as used in multiplexed imaging, often incorporates a *counterstain* to visualize cell nuclei, in addition to the specific cellular markers targeted by the fluorophore-conjugated antibodies. One such counterstain is DAPI (4',6-diamidino-2-phenylindole), which binds to DNA and fluoresces blue when exposed to ultraviolet light. The inclusion of DAPI enables the localization of individual cells within the tissue sample and provides a reference for the distribution and organization of various cell types.

In the example shown in Figure 2-3b, DAPI is used to stain cell nuclei, while the other six channels target specific cell surface markers (Cytokeratin, CD3, CD68, CD8, CD56, CD20) using fluorophore-conjugated antibodies. This allows for the simultaneous visualization of multiple cell types and their interactions within the same tissue section. DAPI staining helps



(a) Hematoxylin and eosin stained section showing ductal carcinoma in situ (pre-cancer). Image from [24].

(b) Multiplex image from section stained with the Opal 7 Kit showing triple-negative breast cancer [4].

Figure 2-3: Histological images of breast tissue stained for bright-field and fluorescence microscopy. Hematoxylin and eosin stains result in the common pink and purple colored images as seen in **(a)**, but other bright-field stains exist for various purposes and tissue types. Note that the image shown in **(b)** is a composite image of seven monochrome channels (DAPI, Cytokeratin, CD3, CD68, CD8, CD56, CD20, see also Appendix A) that have been colored and combined to be displayed as a 3-color channel (i.e. RGB) image.

distinguish individual cells and assess their spatial organization, while the other stains target antigens presented on the cell surfaces, providing insights into the immune landscape of the tissue sample. Table A-1 in Appendix A provides a list of the specific fluorophores used as part of the Opal 7 staining kit [25].

2-1-3 Tumor Immunology and Cellular Markers

This thesis is concerned with the analysis of multiplex immunofluorescence (MxIF) images such as the one shown in Figure 2-3b. (Please refer to Appendix A for a detailed explanation about the individual channels and how they make up the entire image.) The images themselves feature TNBC tissue, with the chosen fluorophores highlighting cells of interest for this analysis. The purpose of this section is to give the reader a brief overview of the relevant concepts and terminology related to tumor immunology and tumor microenvironment (TME) analysis.

Tumor immunology and the TME

Tumor immunology is the study of the complex interplay between the immune system and cancerous cells within the body [26]. The immune system is responsible for identifying and getting rid of abnormal cells, including cancer cells. However, some cancer cells can avoid the immune system and continue to grow, leading to tumor formation. Understanding the relationships between tumor cells and the immune system is essential for creating effective cancer treatments and diagnostics.

The tumor microenvironment (TME) refers to the area surrounding a tumor, which includes not just the cancer cells but also various support cells, immune cells, blood vessels, and the extracellular matrix [26, 27]. The TME plays a significant role in tumor development, progression, and response to therapy. Various factors within the TME can either promote or inhibit tumor growth, and this dynamic environment is a focus of ongoing cancer research [3, 5, 4, 7].

Cellular markers

Cytokeratin (CK) is a family of intermediate filament proteins that are predominantly expressed in epithelial cells [28]. They play a vital role in maintaining the structural integrity of epithelial tissues and are often used as markers to identify and classify different types of epithelial-derived tumors [29]. In the context of tumor biology and cancer research, CK expression can be used to distinguish between cancerous and normal cells or to categorize various tumor subtypes.

Clusters of Differentiation (CD) are cell surface molecules that serve as markers for identifying and characterizing different cell types within the immune system [30]. They are often used as molecular signatures to classify immune cells based on their function, maturation status, and activation state. In the context of tumor immunology, the presence or absence of specific CD markers on immune cells within the TME can provide insights into the immune response and the functional state of immune cells. Additionally, CD markers can be used to identify and classify different types of immune cells within the TME, which can be critical for understanding the immune landscape of tumors and developing targeted therapies.

In this thesis, cellular marker names are used to describe various concepts related to the analysis and classification of the MxIF images in question. When referring to a particular cell itself, it will simply be called by its marker abbreviation, e.g. “CK cells” or “CD8 cells”. When referring to a MxIF image channel, which is intended to capture the cell in question, the word “channel” will be appended in order to reduce ambiguity, e.g. “the signal intensity in the CD3 channel is low compared to the CD8 channel”. When referring to cell labels as they are assigned within the context of machine learning classification, they will be printed in `monospaced font`, include a plus sign (`+` to mark the label’s positivity for the particular cell) and surrounded by quotation marks, e.g. “the classifier performance on `"CK+"` cells was better than those with the `"CD56+"` label”.

2-2 Machine Learning

Machine Learning (ML) is "the study of computer algorithms that improve automatically through experience" [31] and is considered a branch or subset of Artificial Intelligence (AI). Rather than through explicit programming, ML algorithms are expected to perform certain tasks through *learning* from data. A formal definition of this process specifies: "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [31] Typically, the experience E is a data input to the learner. A dataset may be described as "a collection of examples, which are in turn collections of features." [32]

This section provides an introduction to some of the most important concepts in ML. Section 2-2-1 begins by providing an overview of the main learning paradigms in ML. Section 2-2-2 illustrates an example problem and shows two ways to solve it in the context of ML. The example is concluded in Section 2-2-3, where several important model evaluation metrics are discussed. Section 2-2-4 introduces the important concepts of over- and underfitting. Section 2-2-5 and Section 2-2-6 explain the basics of the two most common tasks in ML: regression and classification, respectively. Lastly, Section 2-2-7 presents a special type of classification algorithm, the Perceptron, which serves as the basis for neural networks in deep learning.

2-2-1 Overview of the Machine Learning Paradigms

There are three major learning paradigms in ML: **supervised learning**, **unsupervised learning**, and **reinforcement learning** (see Figure 2-4). However, there is often overlap between these paradigms and they can be further subdivided into more specific categories.

In supervised learning, the algorithm learns from **labeled** datasets (experience E) and its objective is typically to predict the unknown label in previously unseen data (task T), for example to determine whether a picture contains a cat or a dog, or predict a patient's clinical outcome based on measured biomarkers. Performance measures P might include **accuracy**, **precision**, **recall**, **F1-score**, or others depending on the specific problem.

In unsupervised learning, data is provided without labels (experience E) and the objective is to extract useful information from this data such that more insight can be gained from new and unseen data. Typical tasks (T) are **clustering**, which is essentially the assignment of labels to data based on observable characteristics, and **dimensionality reduction**, which puts the data into a representation of lower dimension such that it can be analyzed visually or via other algorithms that do not perform well on high-dimensional data. Performance measures P might include metrics like silhouette score for clustering or reconstruction error for dimensionality reduction.

Semi-supervised learning describes the overlap between supervised and unsupervised learning: In general, a large amount of unlabeled data is used to learn features and patterns present in the data (experience E). This provides "context" to the smaller amount of labeled data, which can then be used to learn aforementioned supervised tasks more easily.

Self-supervised learning is a method where an algorithm learns from unlabeled data by generating its own labels from the data available (experience E). For example, given an

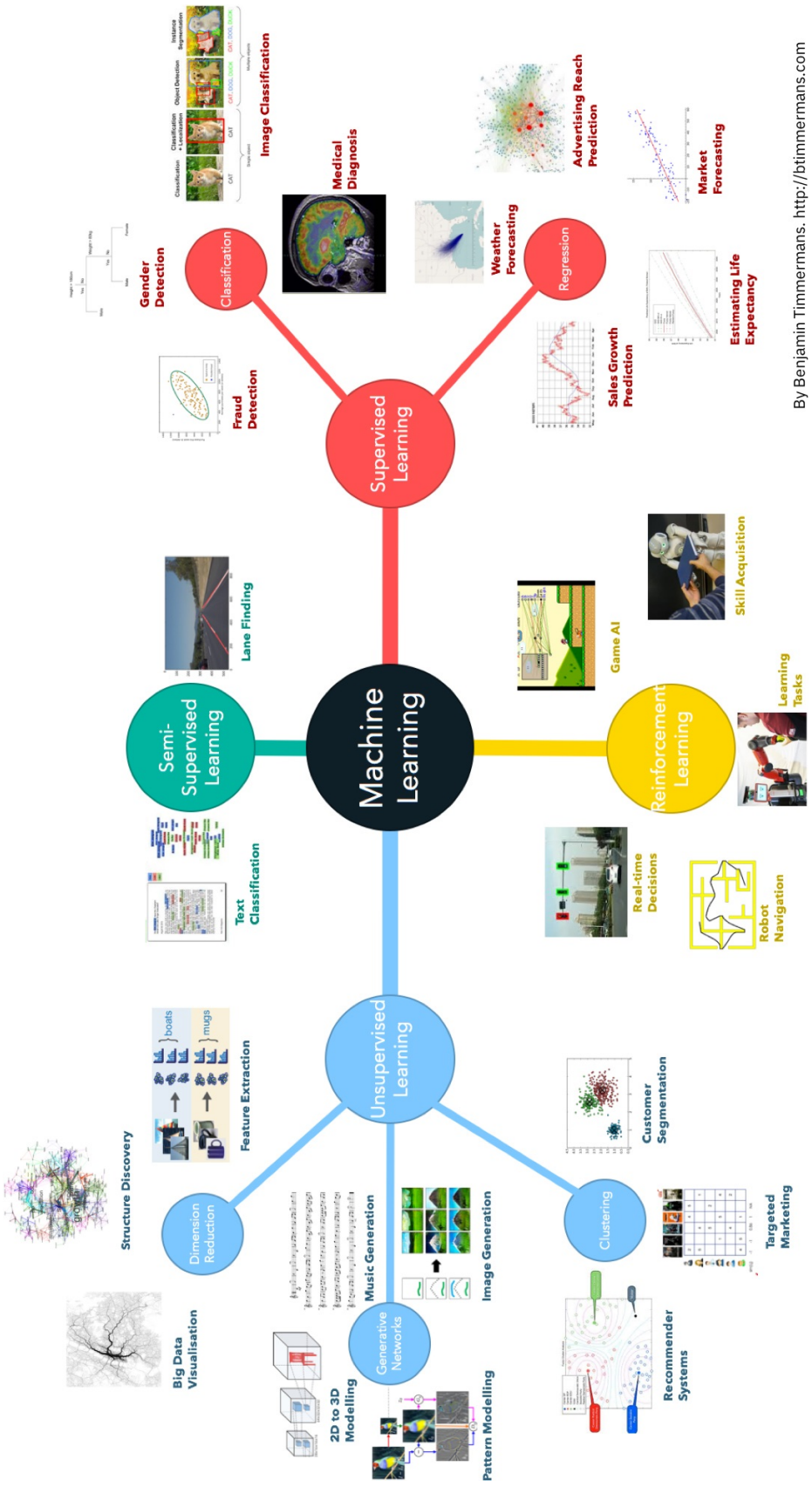


Figure 2-4: Overview of the major learning paradigms in machine learning and the tasks commonly associated with them. In general, supervised learning refers to learning from labeled examples and unsupervised learning refers to learning from unlabeled examples. Semi-supervised learning is a paradigm that combines these two. Reinforcement learning enables learning in live-environments and borrows techniques from other learning paradigms. Image from [33]

unlabeled image, the algorithm may cut out a section of the image and learn to reconstruct it from the remaining information (task T). The cut out section now serves as a label to measure its performance P using metrics such as reconstruction error or similarity scores. Essentially, self-supervised learning is an application of unsupervised learning where an algorithm can be trained in a supervised fashion.

Lastly, reinforcement learning describes the application of machine learning in live environments where the algorithm obtains feedback based on its actions. Typical applications include self-driving cars and game-playing AI. Since we are concerned with the analysis of microscopy images that have already been gathered, we will not consider reinforcement learning in further detail.

A visual example of the aforementioned supervision paradigms can be found in Appendix E.

2-2-2 Practical Example: Prediction of Student Grades

In this section, we will walk through a practical example that illustrates the use of basic ML techniques. The example will be concluded in Section 2-2-3, where important evaluation metrics for classification tasks are presented.

Consider a class of 50 students who took an exam and obtained grades between 1 and 10. Shortly after the exam, the students filled out a survey where they indicated how many hours per week they spent studying for the exam. As shown in Figure 2-5, we can plot the self-reported hours spent studying against the grades obtained in a scatter plot and fit a line on the distribution using **regression**[1], giving us an estimate for the relationship between studying time and exam grades. Using this estimate, we may be able to predict the grades of students who will take the same exam in the future, allowing us to identify students who may be at risk of failing the exam. Referring back to the given definition of ML, the dataset of exam grades/studying time may be considered the experience E, linear regression the task T, and the minimization of the squared residuals the performance measure P.

What happens if we do not have numerical grades available, but only grades specified by "pass" or "fail"? This is an example of a **classification** problem, where we are interested in predicting one or multiple discrete classes. This task can be solved by training a **logistic regression** model: if we consider every numerical grade below 6 a failing grade[2] (0) and every numerical grade of 6 and above a passing grade (1), we can plot the new distribution and fit a sigmoid that models the probability of a student passing or failing the exam depending on their studying time.

This is shown in Figure 2-6, which also includes three decision thresholds for 10 %, 50 %, and 90 %. The point where each of these thresholds (horizontal lines) intersects with the fitted sigmoid, we obtain a classification boundary (vertical lines). Every point to the left of this boundary is assigned the negative class (fail) and everything to the right of this boundary is assigned the positive class (pass).

[1] For this toy example, we can easily obtain a closed-form solution via the ordinary least-squares estimator. However, for more complex datasets or when dealing with non-linear relationships, machine learning techniques such as polynomial regression or nonlinear regression models may be more appropriate.

[2] In this example, we will act as a very strict teacher and won't consider rounding up grades.

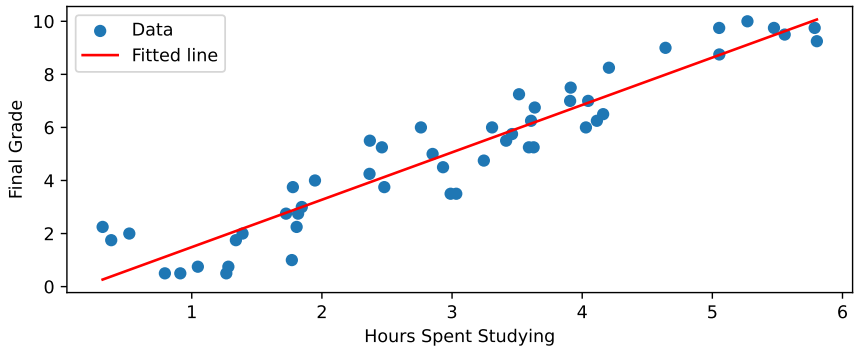


Figure 2-5: Distribution of exam grades obtained by students against their self-reported time spent studying for the exam. The red line is given by the least-squares estimator and provides a model that can predict a student's grade based on their self-reported studying time.

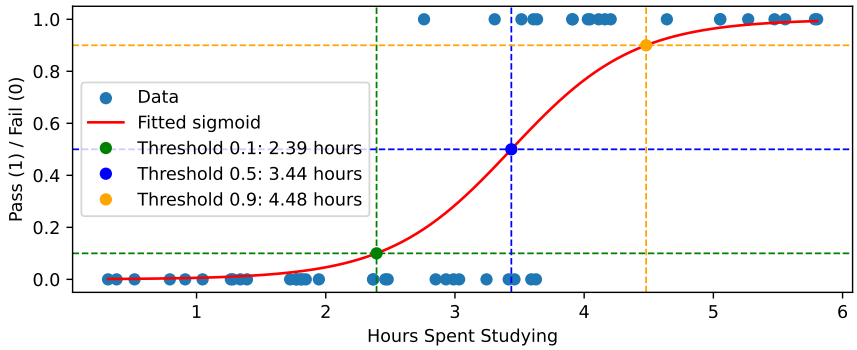


Figure 2-6: The same distribution as shown in Figure 2-5, but with grades converted to pass (1) and fail (0) Here, the red curve was obtained by fitting a logistic regression model, which is an estimator for the probability that a student passed (or failed), given the time spent studying. In order to obtain discrete values from these probabilities, a decision threshold can be placed (given by the horizontal dashed lines) and everything to the left of the obtained boundary (vertical dashed lines) will be assigned fail whereas everything to the right will be assigned pass.

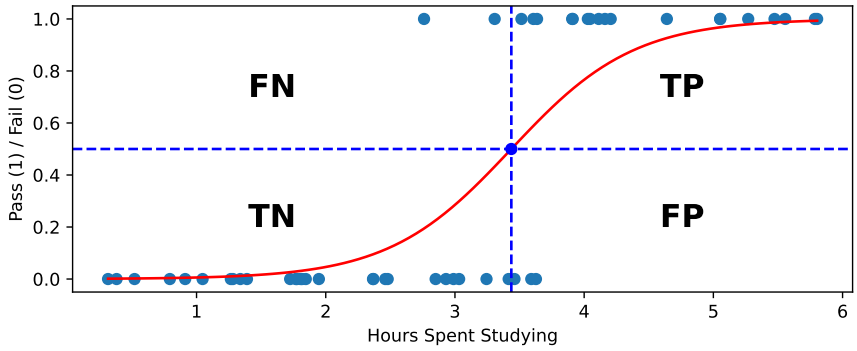


Figure 2-7: The same distribution with logistic regression model as shown in Figure 2-6, only with the decision boundary for 0.5 displayed. The chosen decision boundary can be used to divide the distribution into 4 quadrants corresponding to true positives (top right), false positives (bottom right), true negatives (bottom left), and false negatives (top left).

We have considered a new task T (classification), but the prospect of choosing between various decision thresholds raises the question about an appropriate performance measure P. This will be investigated further in the following section.

2-2-3 Model Evaluation Metrics

For the logistic regression model trained for estimating the probability of passing an exam based on studying time, we may pick a threshold $t = 0.5$ and realize that the plot can be divided into four quadrants as shown in Figure 2-7:

- The top right contains all students who are predicted to pass the exam and actually passed the exam: true positive (TP)
- The bottom right contains all students who are predicted to pass the exam but actually failed the exam: false positive (FP)
- The bottom left contains all students who are predicted to fail the exam and actually failed the exam: true negative (TN)
- The top left contains all students who are predicted to fail the exam but actually passed the exam: false negative (FN)

These are common concepts in statistical classification and are essential for evaluating the performance of a predictive model. Note that these are dependent on the selected decision threshold: As we slide the boundary in Figure 2-7 to the left and right, the amount of TP, FP, TN, and FN predictions will vary. Let $t \in [0, 1]$ be the decision threshold and we can derive the following performance metrics:

Accuracy is the proportion of correct predictions (both positive and negative) made by the model compared to the total number of predictions:

$$\text{Accuracy}(t) = \frac{\text{TP}(t) + \text{TN}(t)}{\text{TP}(t) + \text{TN}(t) + \text{FP}(t) + \text{FN}(t)} \tag{2-1}$$

Precision is the proportion of true positive predictions among all positive predictions made by the model:

$$\text{Precision}(t) = \frac{\text{TP}(t)}{\text{TP}(t) + \text{FP}(t)} \tag{2-2}$$

Specificity is the proportion of true negative predictions among all negative predictions made by the model:

$$\text{Specificity}(t) = \frac{\text{TN}(t)}{\text{TN}(t) + \text{FP}(t)} \tag{2-3}$$

Recall is the proportion of true positive predictions among all actual positive instances in the dataset:

$$\text{Recall}(t) = \frac{\text{TP}(t)}{\text{TP}(t) + \text{FN}(t)} \tag{2-4}$$

While accuracy is a good metric when dealing with a **balanced dataset**, i.e. positive and negative classes are distributed roughly equally, it can be misleading when datasets are **imbalanced**. For example, consider the scenario where 99.9 % of the samples in our dataset have a positive class label and only 0.1 % a negative class label. In this case, we can obtain a classifier with 99.9 % accuracy by always predicting the positive class label. Whether this would be a good classifier or not depends on the application, but this example highlights the fact that we may sometimes want to put more importance on correctly classifying a minority class rather than achieving overall high accuracy.

Measures such as precision, recall, and specificity put importance on minimizing certain types of classification errors. In some applications, such as medical tests or cancer screenings, the trade-off between minimizing false positives and false negatives is crucial. In these cases, the primary concern is identifying as many true positive instances as possible (high recall), while also minimizing false positives (high precision). For example, in cancer screening tests, a high false positive rate may lead to unnecessary invasive procedures, while a high false negative rate may lead to missed cancer cases.

The **F1-Score** is a metric that is often used in machine learning to strike a balance between precision and recall, defined as their harmonic mean:

$$\text{F1-Score}(t) = 2 \cdot \frac{\text{Precision}(t) \cdot \text{Recall}(t)}{\text{Precision}(t) + \text{Recall}(t)} \tag{2-5}$$

By considering both precision and recall, the F1-Score provides a more balanced assessment of the model's performance, particularly in cases where one type of error is more important than the other or when the dataset is imbalanced. In such scenarios, the F1-Score serves as a better performance metric compared to accuracy, which might not fully capture the true effectiveness of the classifier.

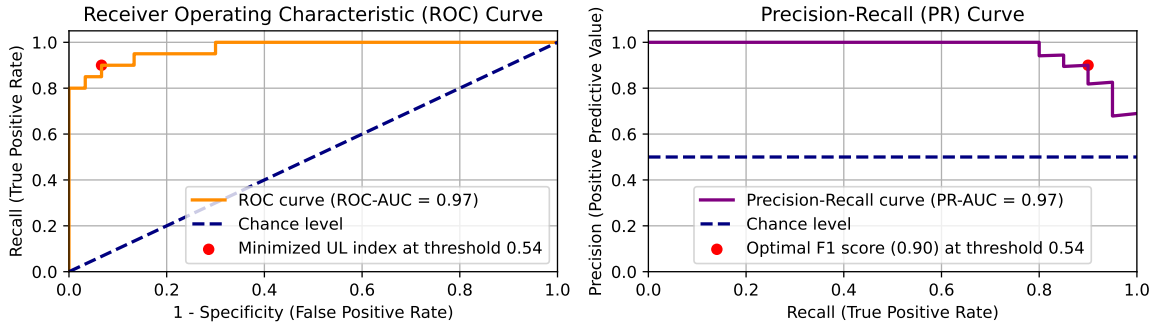


Figure 2-8: Receiver Operating Characteristic (ROC, left) and Precision-Recall (PR, right) curves for the exam pass/fail classifier. A good ROC curve approaches the top left corner of the graph as closely as possible, whereas a good PR curve should approach the top right corner. In both graphs, the dashed line denotes the "chance level", which corresponds to a classifier that only makes random guesses. Both ROC and PR curves can be used for threshold tuning: For the ROC curve, the Upper Left (UL) index is used to determine the optimal decision theshold. Due to its definition, the F1-score is typically used to determine the best threshold in a PR curve. For both curves, the Area Under the Curve (AUC), provides a performance measure of the classifier that does not depend on threshold tuning.

The aforementioned metrics depend on the decision threshold and may therefore not be ideal if we want to compare the performance of various classifiers with each other. To address this issue, we can make use of plots such as the **Receiver Operating Characteristic (ROC)** curve (shown on the left in Figure 2-8) and the **Precision-Recall (PR)** curve (shown on the right in Figure 2-8).

The ROC curve is a plot of Recall (also known as True Positive Rate) against $1-$Specificity (also known as the False Positive Rate) by varying the threshold from 0 to 1 (left plot in Figure 2-8). The dashed diagonal line represents the chance level, i.e., a classifier that makes random guesses. The ROC curve can be used to determine a decision threshold: the red dot indicated in the upper left corner is the point where the Upper Left (UL) index[3] is minimized. The Area Under the Receiver Operating Characteristic Curve (ROC-AUC) is a **performance measure that does not depend on the threshold**, making it a suitable metric for comparing the performance of different classifiers.

Similarly, the PR curve is a plot of Precision (also known as Positive Predictive Value) against Recall by varying the threshold from 0 to 1 (right plot in Figure 2-8). Here, the dashed horizontal line represents the chance level, i.e., a classifier that makes random guesses. The PR curve can be used to find a threshold based on the maximum F1-score, which is indicated by the red dot in the upper right corner. In this case as well, the Area Under the Precision-Recall Curve (PR-AUC) is a **performance measure that does not depend on the threshold**. The PR-AUC is considered to be a better performance measure in cases dealing with imbalanced datasets [34].

This concludes our practical example of applying ML techniques to fit models on a distribution of students' exam grades and their self-reported studying time and, in the case of a binary

[3]The UL index is just one example of numerous other indices that can be considered. The choice of an appropriate index depends on the specific application and the goals of the analysis.

classifier, evaluating these models. According to Figure 2-8, both the UL index as well as the maximum F1-score tell us that the optimal decision threshold for our binary classifier is 0.54 (note that these are not always the same, necessarily). An important aspect of training ML models that we did not consider yet is the separation of data into **training sets** and **test sets**, which helps to prevent overfitting and underfitting, ensuring that our models generalize well to new, unseen data. The following sections will address these concepts, as well as provide more formal definitions of the regression and classification techniques that were used to obtain the models in this example.

2-2-4 Overfitting, Underfitting, and Generalization

In the previous sections, we introduced a simple example that illustrates regression and classification in Section 2-2-2 and investigated performance metrics in Section 2-2-3. It is crucial to understand the concepts of overfitting, underfitting, and generalization, which are key aspects to consider when building and evaluating machine learning models. These concepts are closely related to the **training and test sets** and the model's **capacity** to fit the data. Understanding these ideas will help us choose the right model architecture, training strategies, and evaluation techniques to ensure our models are robust and effective in handling unseen data.

Essentially, a ML algorithms learns by (repeatedly) solving optimization problems. However, minimizing the **training error**, that is the error observed during training, is not of primary concern: since we are interested in the learning algorithm performing well on unseen data, a portion of the available data is withheld during training so it can be used to **test** the model afterwards. The two resulting sets of this split are referred to as the **training set** and **test set**. The error observed when testing the model against the test set is called the **generalization error** and is a measure of how well the model is expected to perform on unseen data. In general, the training error being too high is a sign of **underfitting** and the training error being much lower than the generalization error is a sign of **overfitting**. The measure of a model's fitting capability is referred to as **capacity**: a model with low capacity tends to underfit and a model with high capacity tends to overfit.

One way to alter a model's capacity is by increasing or decreasing its complexity. Consider the task of fitting a curve to n data points that are randomly drawn from a 2^{nd} order polynomial

$$y = w_0 + w_1 x + w_2 x^2 \tag{2-6}$$

where (x, y) is a pair of coordinates and w_i the unknown coefficients of the polynomial to be determined. Let the solving model be of the form

$$\mathbf{y} = \mathbf{X}\mathbf{w} \tag{2-7}$$

where $\mathbf{y} \in \mathbb{R}^{n\times 1}$ is the observation vector, $\mathbf{X} \in \mathbb{R}^{n\times m}$ the data matrix, and $\mathbf{w} \in \mathbb{R}^{1\times m}$ a vector of weights to be determined. The data matrix has the form

$$\mathbf{X} = \begin{bmatrix} \mathbf{1} & \mathbf{x} & \mathbf{x}^2 & \dots & \mathbf{x}^{m-1} \end{bmatrix} \tag{2-8}$$

where $\mathbf{1}$ is a $n \times 1$ ones vector and $\mathbf{x} \in \mathbb{R}^{n\times 1}$ the data vector corresponding to $\mathbf{y}$. Clearly, the optimal choice for the model order should be $m = 3$. As long as $n \geq m$, the solution is then

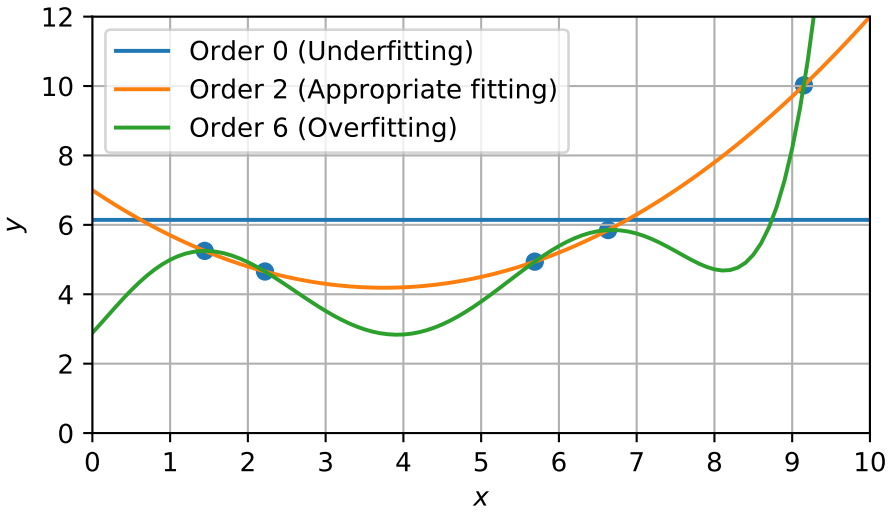


Figure 2-9: Example of underfitting, overfitting and appropriately fitting a least squares solution to a 2nd order polynomial by varying the solution polynomial's order. The blue curve simply represents the mean of all points but does not represent the underlying model accurately. The green curve, although hitting all the points, also does not seem to represent the underlying model. The orange curve appears to match the underlying model perfectly.

given by the ordinary least squares solution [35]

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{2-9}$$

Figure 2-9 shows the results of choosing models with $m \in \{1, 3, 7\}$ and $n = 5$. For $m = 1$ we can see underfitting and $m = 7$ (where the pseudo-inverse was used to compute the solution) overfitting. Although the $m = 7$ model is perfectly fitting through every point given (the training set), we can clearly see that any additional points (as part of a test set) would not fall on its curve. In this example, $m = 3$ is the optimal model capacity since it minimizes generalization error.

Figure 2-10 shows a typical relationship between training error, generalization error, and the location of optimal capacity. In our previous example, the underfitting zone would correspond to $m < 3$ and the overfitting zone to $m > 3$.

Altering the model complexity in order to minimize generalization error is not always feasible, desirable, or possible. In the least squares case, the available data may result in a (nearly) singular $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ matrix which may lead to convergence on an infeasible or unrealistic solution. For example, we may have more knowledge about a data-generating model than the obtained data shows. In this case, we may be able to add more information to the cost function to be minimized by means of **regularization**. In the case of ordinary least squares, which minimizes the cost function $||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2$, we may add a **weight decay** term such that the cost function becomes

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \mathbf{w}^{\mathrm{T}}\mathbf{w} \tag{2-10}$$

where λ is the regularization weight. The solution is then given by

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{2-11}$$

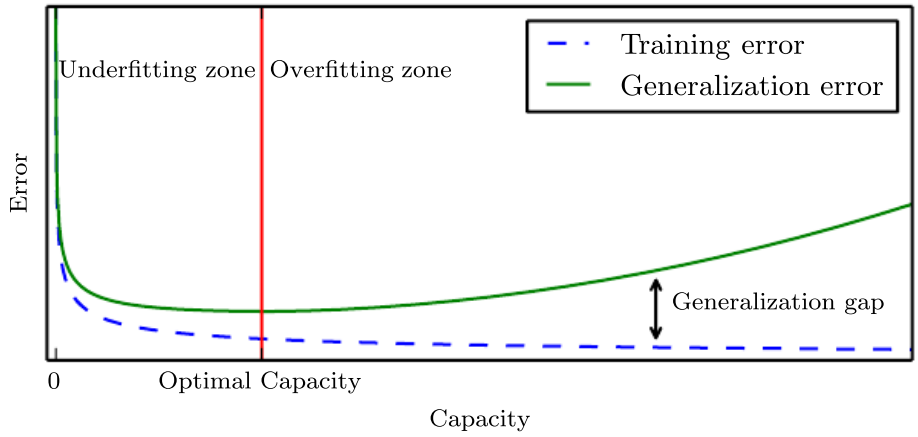


Figure 2-10: Typical relationship between model capacity and error. The model's optimal capacity is where the generalization error is lowest, that is, the error when it is applied to data it has not seen during training. Although the training error can be minimized further than the generalization error, doing so typically leads to overfitting as the model learns to remember examples from the training data and fails to properly predict unseen data. Image from [32].

where $\mathbf{I}$ is the identity matrix. By tuning λ, we determine the trade-off between minimizing the least squares solution and keeping the magnitude of the weights low, which can help to mitigate bad training results if the learner encounters an abundance of "bad data". The regularization term added in Eq. (2-10) is also known as L2 regularization and a multitude of other regularization methods exist which may be chosen depending on the task to be learned.

2-2-5 Regression and Regularization

Regression tasks are one of the primary problems in supervised learning (see Section 2-2-2 where treated a simple example). The goal of regression is to approximate an unknown function with only its inputs and outputs known. Importantly, these are functions with continuous (non-discrete) outputs. Recall the example in Section 2-2-4 and Figure 2-9 of fitting polynomials of varying degrees to a set of points. Although the approximated functions are not linear functions themselves, they are linear with respect to the parameters to be estimated and can be expressed as in Eq. (2-7), which allows us to perform **linear regression**. Commonly, regularization terms are used in order to prevent overfitting as shown in Eq. (2-10). A more general cost function can be written as

$$J(\boldsymbol{\theta}) = \mathbf{e}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_p \tag{2-12}$$

where $\mathbf{e}(\boldsymbol{\theta})$ is the error to be minimized, $\boldsymbol{\theta}$ the minimizing variable vector (previously expressed as $\mathbf{w}$), and p the order of regularization. The most commonly used norms are with $p = 1$ (**lasso regression**) and $p = 2$ (**ridge regression**) which are graphically shown in Figure 2-11: The regularization terms force the solution closer to the origin by requiring them to be contained within the red areas. Lasso regression tends to force sparse solutions, i.e. solutions where one or several elements of the solution vector are 0, and is useful for performing feature selection, for example if variables are suspected to be highly correlated. Note that, as $\lambda \to 0$

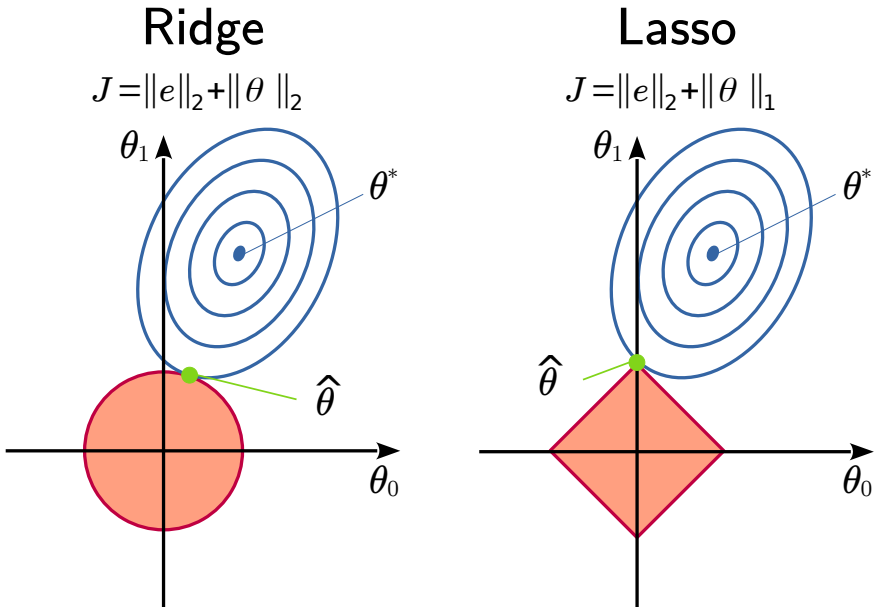


Figure 2-11: Visualization of two commonly applied regularization terms. The optimal solution without regularization $\boldsymbol{\theta}^*$ is given by the point surrounded by the blue contour curves. Regularization forces the solution closer to the origin by finding the intersection between one of the contour curves and the area defined by the regularization term. Ridge regression forces solutions closer to the origin whereas lasso regression typically forces solution on one of the axes, leading to a sparser vector representation of the solution vector $\hat{\boldsymbol{\theta}}$. Image adapted from [36].

in Eq. (2-12), the size of the red areas in Figure 2-11 tends towards infinity, returning the non-regularized solution.

2-2-6 Discriminative Classification

Another one of the primary problems in supervised learning is classification (see Section 2-2-2 where treated a simple example). The goal is to assign class labels to unseen data with only knowledge of data and their labels obtained from the same distribution source (ideally, the test data and training data are independent and identically distributed (i.i.d.)). Assuming the data is drawn from distribution $p(\mathbf{x})$, there is a (posterior) probability $p(y|\mathbf{x})$ linking the class label y to the data $\mathbf{x}$. A simple formulation for a binary classifier (two classes) can then be given by

$$\begin{aligned} \text{if } p(y_1|\mathbf{x}) > p(y_2|\mathbf{x}), \quad & \text{assign to } y_1. \\ \text{Otherwise}, \quad & \text{assign to } y_2. \end{aligned}$$

For k classes, we can express the posterior probability using Bayes' theorem

$$p(y_k|\mathbf{x}) = \frac{p(\mathbf{x}|y_k)p(y_k)}{p(\mathbf{x})} \tag{2-13}$$

relating it to the class conditional distribution $p(\mathbf{x}|y_k)$, the class prior probability $p(y_k)$, and the unconditional data distribution $p(\mathbf{x})$. In this case, the **Bayes classifier** would compute

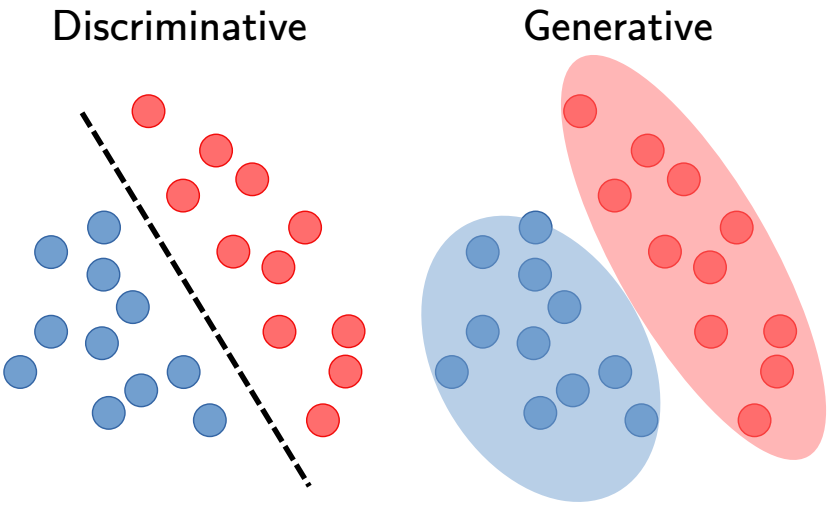


Figure 2-12: Visualization of discriminative and generative classification. Discriminative methods attempt to separate the data by drawing boundaries between them such that new data can be classified depending on where they land with respect to the drawn boundaries. Generative models attempt to model the underlying distribution of the data such that new data can be assigned a probability of belonging to any class. Image adapted from [37].

the maximum a posteriori probability (MAP) with respect to k and assign a point to class y_k respectively.

In practice, $p(\mathbf{x})$ is given by the available data and $p(y_k)$ can be estimated from it. The class conditional distribution, however, is not straightforward to determine. Two approaches to solve this are **discriminative classification** and **generative classification**. A visual example of the difference is shown in Figure 2-12: Discriminative classifiers directly estimate the boundaries between classes, whereas generative classifiers model the underlying probability distributions of the data in order to determine class labels of new data. In the following, we will focus on discriminative classifiers.

Discriminative classifiers model the posterior probability distribution directly as $f(\mathbf{x}) = \hat{p}(y_k|\mathbf{x})$. A common discriminative classifier is the **logistic classifier** which makes use of performing **logistic regression** on the input data (this is the same type of classifier we have used in Section 2-2-2). For binary classification, assume we can model the log odds ratio of the two posteriors as an affine function

$$\ln\left(\frac{\hat{p}(y_1|\mathbf{x})}{\hat{p}(y_2|\mathbf{x})}\right) = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}. \tag{2-14}$$

With $\hat{p}(y_1|\mathbf{x}) = 1 - \hat{p}(y_2|\mathbf{x})$ and taking the exponent of both sides, we obtain

$$\hat{p}(y_2|\mathbf{x}) = f_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{1 + \exp(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})} \tag{2-15}$$

which is a logistic function with parameter vector $\boldsymbol{\theta}$ to be determined. By setting a threshold to the value of $\hat{p}(y_2|\mathbf{x})$, we can make a decision between assigning to class y_1 or y_2. The logistic loss function is its negative log-likelihood function

$$J(\boldsymbol{\theta}) = -\sum_i \Big(y_i \log\left(f_{\boldsymbol{\theta}}(\mathbf{x}_i)\right) + (1 - y_i) \log\left(1 - f_{\boldsymbol{\theta}}(\mathbf{x}_i)\right)\Big) \tag{2-16}$$

where $\mathbf{x}_i$ refers to the ith point in the dataset and $y_i \in \{0, 1\}$ to its respective label. This is also known as the **binary cross-entropy loss function** and is a commonly used loss function in machine learning and deep learning models for binary classification tasks. Although $J(\boldsymbol{\theta})$ is convex and its derivative can be used to find the global minimum with respect to $\boldsymbol{\theta}$, no closed form solution exists. Therefore, the **gradient descent** algorithm is used in order to vary the elements of $\boldsymbol{\theta}$ and find the solution:

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \ldots, \theta_n) \tag{2-17}$$

where θ_j is jth element of $\boldsymbol{\theta}$ and η is the learning rate determining by how much θ_j will vary per iteration.

2-2-7 Perceptron Algorithm for Classification

An important discriminative classifier which forms the basis for neural networks is the **perceptron**. Similar to logistic regression, we assume that the data can be separated by the hyperplane

$$\boldsymbol{\theta}^{\mathrm{T}} \mathbf{x} = 0 \tag{2-18}$$

with the difference that the decision boundary is learned on the data directly without the use of regression. To this end, we define the label for point $\mathbf{x}_i$ in our dataset as $y_i = \pm 1$ and furthermore that

$$\begin{aligned} &\text{if } y_i = +1, \quad \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}_i > 0, \\ &\text{if } y_i = -1, \quad \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}_i < 0. \end{aligned}$$

These conditions can be combined into

$$-y_i \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}_i < 0 \tag{2-19}$$

which holds true whenever a point is correctly classified. In order to optimize the perceptron's performance, we are only interested in the points it incorrectly classified, leading to the formulation of the cost function

$$J(\boldsymbol{\theta}) = \sum_{i \in Y} y_i \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}_i = \sum_i \max(0, -y_i \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}_i). \tag{2-20}$$

where Y is the set of indices of all misclassified datapoints. This cost function has the trivial solution $\boldsymbol{\theta} = \mathbf{0}$ but is otherwise optimally solved when $J(\boldsymbol{\theta}^*) = 0$. Its derivative is

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \ldots, \theta_n) = \sum_{i \in Y} y_i x_{i,j} \tag{2-21}$$

and can be substituted in the gradient descent update equation Eq. (2-17) to obtain

$$\theta_j \leftarrow \theta_j - \eta \sum_{i \in Y} y_i x_{i,j}. \tag{2-22}$$

A disadvantage of this update algorithm is that it does not converge if the dataset $\mathbf{x}$ is not linearly separable. If the dataset is linearly separable, however, it has the advantageous

property of being able to perform updates *on-line*, that is, without considering the entirety of the dataset in its update calculation, while still converging on the optimal solution. This is in contrast to the gradient descent update equation used for optimizing the logistic (or any other) regression cost function, which requires the entire dataset in its computation in order to find the optimal solution.

The ability to perform the gradient descent optimization using only a **minibatch** of data points randomly sampled from the entire dataset is called **stochastic gradient descent** (treated in more detail in Section 2-3-3) and is an important algorithm since it reduces computation time when training models on very large datasets.

The Perceptron algorithm forms the basis for more complex machine learning models, such as neural networks used in deep learning architectures, which will be explored in the following section.

2-3 Deep Learning

Deep Learning (DL), as a sub-field of Machine Learning (ML), comprises learning algorithms making use of artificial neural networks (ANNs) — usually simply called **neural networks (NNs)**. Below, the basic steps to train a neural network are summarized. The short explanations of every step provide this section's outline (for clarity, the sections that will introduce new concepts are printed in **bold face**).

1. Input the training example(s): The first step is to input a training example into the neural network. Very simply, the input could just be a vector (for example: a vector containing the student-reported hours spent studying for an exam as shown in Section 2-2-2).
2. Forward propagation: The input is passed through the neural network one layer at a time, with each layer transforming the input into a higher-level representation by means of an activation function. **The necessity of layers and activation functions is presented in Section 2-3-1**.
3. Compute the loss: The predicted output is then compared to the true output (i.e., the labels — in the example of Section 2-2-2, those were the predicted exam grades) using a loss function. The loss function depends on the task, with some examples already provided in Section 2-2-5 and Section 2-2-6.
4. Backpropagation: Once the loss is computed, the gradients of the loss with respect to the weights of the network are computed using backpropagation. **Backpropagation is an algorithm for efficiently computing these gradients and is presented in Section 2-3-2**.
5. Update the weights: The gradients of the loss with respect to the network's weights are then used to update the weights of the network using an optimizer. With gradient descent (Section 2-2-7), we have already seen a simple optimizer for weight updates. **Some advanced optimizers that are used in training neural networks are presented in Section 2-3-3**.
6. Repeat: The previous steps are repeated until the network stops improving or an acceptable performance is achieved.

Neural networks are not limited to one-dimensional inputs, but can handle higher-dimensional inputs such as images. In these cases, specialized types of neural networks called CNNs are used. **Important concepts of CNNs for image recognition tasks are presented in Section 2-3-4**.

2-3-1 Extending the Perceptron: Multi-layer Perceptrons and Activation Functions

In Section 2-2-7, we have shown the perceptron classifier. In the context of DL, a single perceptron, or several perceptrons acting in parallel, are referred to as single-layer perceptron (SLP) networks due to them constituting a single layer of calculation units between input and

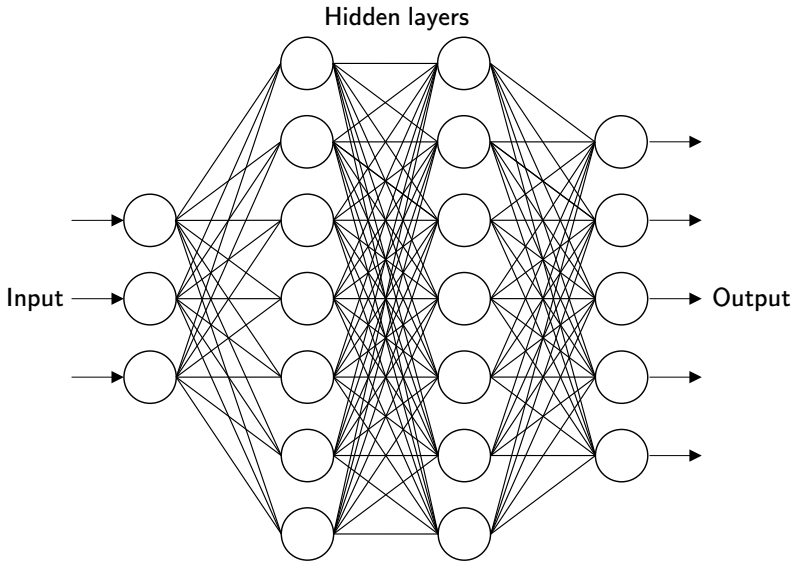


Figure 2-13: Schematic of a multilayer perceptron — a simple type of neural network. The network has an input layer of size 3, an output layer of size 5, and two hidden layers of size 7 each. Since every neuron of one layer is connected to every neuron of the next layer, this is also an example of a fully-connected network.

output. This is in contrast to multilayer perceptrons (MLPs), with a simple example provided in Figure 2-13, where multiple layers of perceptrons are placed in series. These additional layers that act between the input layer and the final output layer are called **hidden layers**. The MLP is a simple example of a **deep neural network** where the amount of hidden layers determines its depth. Individual perceptron nodes are often referred to as **neurons**.

Figure 2-14 graphically shows the working principle of a perceptron: The sum of weighted inputs is passed to an **activation function** which produces the perceptron's output. Algebraically, we can write

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x}) \tag{2-23}$$

where ϕ is the activation function, $\mathbf{W}$ the weight vector (or matrix), $\mathbf{x}$ the input vector, and $\mathbf{y}$ the output vector. The activation function marks an important difference between perceptrons (or neurons) within an MLP and the perceptron classifier as seen previously. Whereas the perceptron classifier uses a step function, which outputs either -1 or $+1$, the neurons in MLPs may utilize a wide range of nonlinear activation functions.

The necessity for nonlinearity of the activation function can be demonstrated by a simple example: Consider a MLP with a single hidden layer such that

$$\begin{aligned}\mathbf{y} &= \phi(\mathbf{W}_1\mathbf{x}) \\ \mathbf{z} &= \phi(\mathbf{W}_2\mathbf{y}) \\ \mathbf{z} &= \phi(\mathbf{W}_2\phi(\mathbf{W}_1\mathbf{x}))\end{aligned}$$

where $\mathbf{y}$ is the output of the hidden layer and $\mathbf{z}$ is the output of the output layer. With a linear activation function $\phi(x) = x$ we obtain

$$\mathbf{z} = \mathbf{W}_2\mathbf{W}_1\mathbf{x} = \mathbf{V}\mathbf{x} \tag{2-24}$$

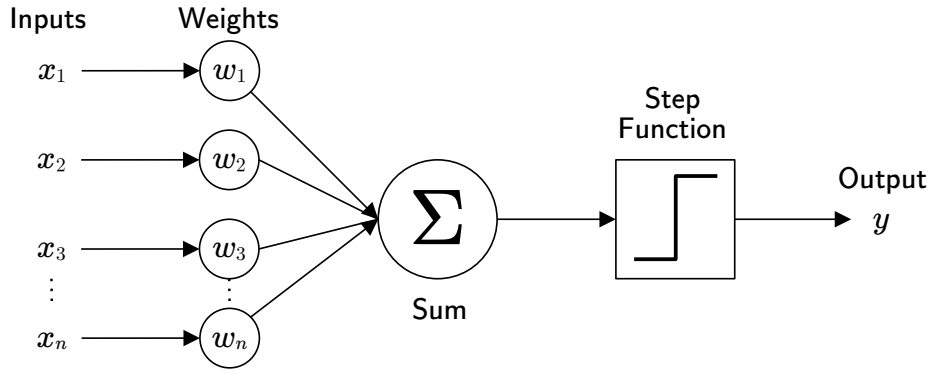


Figure 2-14: Schematic overview of Rosenblatt's Perceptron. The inputs $\mathbf{x}$ are multiplied with tunable weights $\mathbf{W}$, summed and passed as argument to a step function, whose output is the perceptron's output.

showing that the MLP collapses into a SLP with linear activation function. For nonpolynomial activation functions, however, the universal approximation theorems state that any function f may be arbitrarily well approximated by a neural network with a single hidden layer containing arbitrarily many nodes [38, 39][4].

One of the most commonly used activation functions in neural networks is the Rectified Linear Unit (ReLU) [40]. It can be simply defined as

$$\phi_{\text{ReLU}}(z) = \max(0, z) \tag{2-25}$$

or in words: it returns the positive part of its argument. Many more activation functions exist and an overview of commonly used ones is given in Table F-1 in Appendix F. Depending on the choice of activation function at the output layer, a neural network can perform regression or classification.

The definition of the MLP and activation functions enables a regularization technique that is commonly used in neural networks: **dropout**. Dropout is a method that helps prevent overfitting by randomly setting a fraction of input units within a layer to zero at each training update (e.g., a dropout of 0.2 will disable a random 20 % of neurons within a layer). In essence, during training, dropout can be considered as an activation function for the selected neurons, where:

$$\phi(z) = 0 \tag{2-26}$$

This process encourages the network to learn more robust features, as it cannot rely on the presence of specific neurons and must adapt to the absence of some input information.

2-3-2 Training Neural Networks Efficiently with Backpropagation

The example of the MLP has shown that, mathematically, neural networks are the repeated and cascading application of nonlinear functions to weighted sums. neural networks serve as function approximators allowing for more complexity than the classic ML models for regression and classification. As such, they can be trained by optimization via gradient descent,

[4]For mathematically similar reasons, Fourier series can approximate functions arbitrarily well by summing weighted sinusoidal terms.

but with the added computational challenge of its loss function depending on a large amount of parameters, namely the weight and bias for each neuron. Successful training of a neural network is therefore dependent on efficient gradient calculation, which is facilitated by the **backpropagation algorithm**.

The backpropagation algorithm makes use of four equations (their derivation based on [41] is shown in Appendix C):

$$\boldsymbol{\delta}^{(L)} = \nabla_a J \circ \phi' \left(\mathbf{z}^{(L)} \right), \tag{2-27}$$

$$\boldsymbol{\delta}^{(l)} = \left(\mathbf{W}^{(l+1)} \right)^{\mathrm{T}} \boldsymbol{\delta}^{(l+1)} \circ \phi' \left(\mathbf{z}^{(l)} \right), \tag{2-28}$$

$$\frac{\partial J}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)}, \tag{2-29}$$

$$\frac{\partial J}{\partial b_j^{(l)}} = \delta_j^{(l)}. \tag{2-30}$$

Importantly, $\delta_j^{(l)}$ is the *error* of the jth neuron in layer l and gives an indication as to how much a change in that neuron's value is influencing the loss function J. The backpropagation algorithm then follows:

1. Input: Set one training data sample input $\mathbf{x}$ as the activation of the first layer $\mathbf{a}^{(1)}$.
2. Feedforward: Compute the weighted inputs $\mathbf{z}^{(l)}$ and activations $\mathbf{a}^{(l)}$ for all subsequent layers $l \in \{2, 3, \ldots, L-1, L\}$.
3. Output error: Starting at the output layer L, compute its neurons' errors $\boldsymbol{\delta}^{(L)}$ with Eq. (2-27).
4. Backpropagate the error: Compute the remaining neurons' errors by recursively using Eq. (2-28) moving *backwards* through the network's layers $l \in \{L-1, L-2, \ldots, 3, 2\}$.
5. Output: Use Eq. (2-29) and Eq. (2-30) to compute the gradient of the loss function with respect to every neuron's weight $w_{jk}^{(l)}$ and bias $b_j^{(l)}$.

2-3-3 Optimization Algorithms for Weight Updates

Whereas the backpropagation algorithm can efficiently compute the loss function's gradient with respect to the weights and biases of every neuron, it is the purpose of an optimization algorithm, or simply **optimizer**, to use these gradients to update their values.

Let us re-write the gradient descent update equation (2-17) from Section 2-2-6 with the notation for weights and biases introduced in Section 2-3-2:

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \eta \frac{\partial J}{\partial w_{jk}^{(l)}}, \quad b_j^{(l)} \leftarrow b_j^{(l)} - \eta \frac{\partial J}{\partial b_j^{(l)}}, \tag{2-31}$$

where η is the learning rate and the derivatives of the loss function with respect to the weights and biases are as given in Eq. (2-29) and Eq. (2-30), respectively.

We do not need to compute the loss and its gradients for every example in our input dataset all at once. Instead, we can split the input data into randomly sampled **minibatches** and compute the loss, its gradients and the weight updates in sequence. One complete pass through the entire dataset, covering all minibatches, is referred to as an **epoch**. During the training process, multiple epochs are performed to iteratively refine the model's parameters.

This is called **Stochastic Gradient Descent (SGD)** and is an important algorithm in deep learning due to several key advantages:

- **Online Learning:** SGD's ability to work with mini-batches supports online learning, where the model can be updated continuously as new data becomes available.
- **Computational and Memory Efficiency:** SGD's mini-batch approach allows for faster convergence compared to batch gradient descent, as the model can be updated more frequently with smaller sets of data. This also results in a reduced memory requirement for the optimization process, enabling the training of models on devices with limited memory.
- **Noise-induced Exploration:** The inherent randomness in selecting mini-batches introduces noise into the optimization process, helping the optimizer escape local minima and potentially find better global solutions.

There are numerous advanced optimization algorithms used in modern deep learning algorithms. Two of these, which have been used for training the models as part of this thesis, are explained below.

Adam

The Adam[5] optimizer [42] is an enhancement over the standard SGD technique. It incorporates the advantages of two other well-known optimization techniques, AdaGrad [43] and RMSprop [44], and maintains separate learning rates for each parameter. These learning rates are adaptively updated based on the first and second moments of the gradients. Consequently, Adam provides the following key benefits:

- **Faster Convergence:** Adam converges faster and exhibits reduced oscillations compared to SGD, making it particularly effective for training deep learning models with noisy and sparse gradients.
- **Less Sensitivity to Initial Learning Rate:** The Adam optimizer is less sensitive to the choice of the initial learning rate and more robust to variations in the gradients, which leads to improved convergence properties compared to SGD.

LARS

The Layer-wise Adaptive Rate Scaling (LARS) optimizer [45] is an optimization algorithm that distinguishes itself from the Adam optimizer by employing a separate learning rate for

[5]Not an acronym, but derived from Adaptive Moment Estimation

each layer in the network. This approach addresses the challenge of different layers in the network requiring distinct learning rates to achieve optimal performance. LARS offers the following notable features:

- **Magnitude-based Update Control:** LARS controls the magnitude of the update with respect to the weight norm, which allows for better control of training speed and improved stability during training.
- **Adaptive Learning Rates:** By incorporating a separate learning rate for each layer, LARS can adaptively adjust the learning rates based on the magnitude of the gradients, leading to better convergence properties compared to traditional SGD.
- **Effective for Large-Scale Training:** The layer-wise adaptive rate scaling allows for more effective training of deep learning models with varying gradient magnitudes across layers, making it particularly suitable for large-scale training tasks.

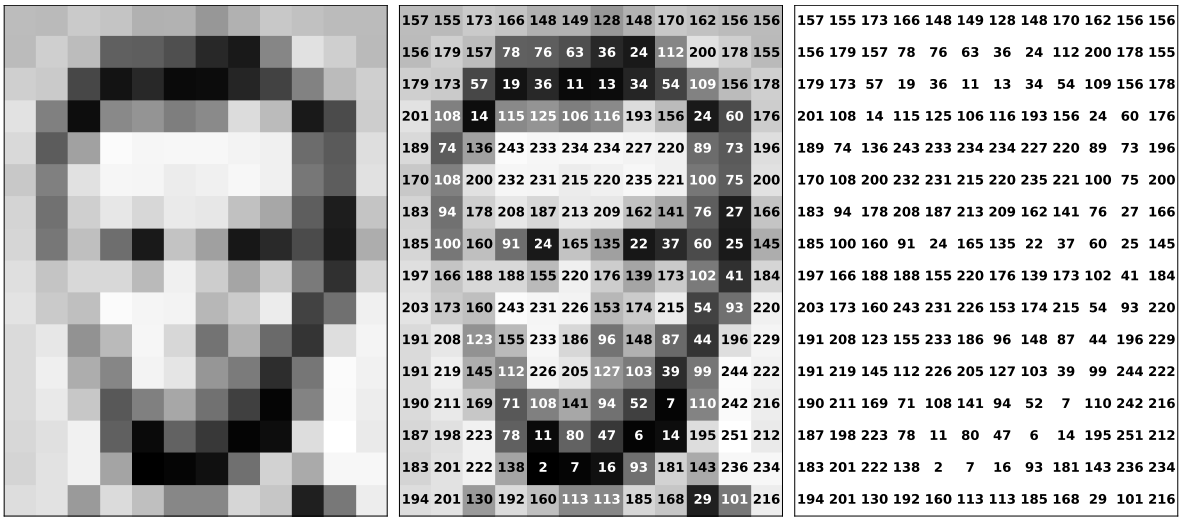
2-3-4 Convolutional Neural Networks (CNNs)

CNNs are a class of deep learning models specifically designed for image recognition and analysis tasks. They are particularly adept at identifying patterns and features within images by exploiting spatial information.

To understand how CNNs process images, let us start with an example: Figure 2-15 shows three representations of a grayscale image. Figure 2-15a shows the image as we would normally see it on a computer screen. All digital images can be represented as an array of pixel brightness values. This is shown in Figure 2-15b, where all pixels in the image are overlaid with their respective brightness value ranging between 0 and 255. For a color image, the image would have three channels (typically corresponding to the intensity values of the colors red, green, and blue) rather than just one, and microscopy images for digital pathology may have even more channels depending on how many different spectral conditions the image has been captured at. Figure 2-15c shows the image information simply as those numerical values, which is closer to how a computer would "see" this image.

In order to train a neural network to recognize and classify this image, it may be tempting to stack the image's pixel brightness values in Figure 2-15c in a tall vector and feed it as input to a simple network like we have seen previously. In Section 2-3-1, we have only considered networks that are **fully connected**, that is, every neuron in layer l is connected to every neuron in layers $l-1$ and $l+1$. However, using a fully connected network for image recognition may not be ideal, as it does not effectively capture spatial information or identify local patterns that make up features like eyes, nose, and mouth. Furthermore, if the pixels in the image were to be shifted slightly in a given direction, the input to the neural network could look radically different despite it essentially representing the same image.

For image recognition, we are therefore interested in robustness to spatial variance as well as the network's ability to perform **feature extraction** such that it can perform classification on a set of high-level features it has detected in an image. The solution to this problem is to use **convolutions** in order to propagate information through the network. Neural networks utilizing convolution are therefore called **convolutional neural networks (CNNs)** or **ConvNets**.



(a) Image representation. **(b)** Added pixel brightness values. **(c)** Matrix representation.

Figure 2-15: Three representations of a grayscale image. The purely visual representation in **(a)** is most useful for human interpretation. For interpretation by a computer, however, it has to be represented purely as a numerical array as shown in **(c)**. Computer image formats typically represent pixel brightness as integer values between 0 (black) and 255 (white), as is shown here, but values ranging between 0 and 1 are also possible. Image adapted from [46].

Convolutions

The application of convolutions is similar to the way they are used in image processing in order to apply sharpening, blurring, edge detection, or other filtering to the image. The way information propagates through a CNN via convolution is shown in Figure 2-16. A convolution filter (also referred to as kernel) is applied to a subset of pixels in the source layer, with the resulting value of this matrix operation being written to a pixel in the destination layer. This operation is repeated by sliding the filter across the source layer until all pixels in the destination layer are populated.

To put this into the context of neural networks as seen previously, it is important to note that the convolution filter contains the weights that are learned during training and that it only connects a subset of neurons (pixels) in a layer with a single neuron in the next layer. After this matrix calculation, a bias term (subject to training) is added to the result and input to that neuron's activation function. Since the result of a convolution is assumed to contain some information about detected features, it is called a **feature map** rather than an image.

Multiple convolutions with different filters may be applied per layer, resulting in the feature map not only having a width and a height, but also a depth which corresponds to the amount of convolution operations.

Pooling

Another important operation within CNNs is **pooling**. An example of pooling is shown in Figure 2-17: A group of pixels in the source layer is combined a to represent a single pixel

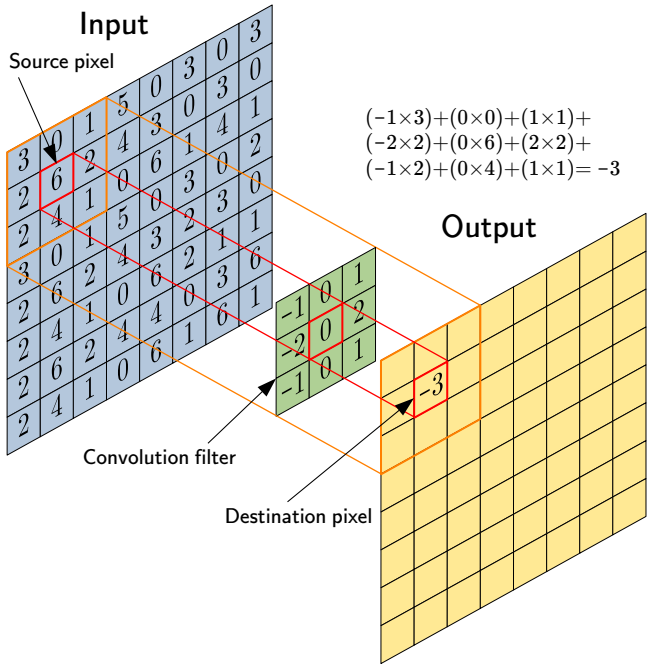


Figure 2-16: Visualization of a convolution operation between two layers. Note that it is the convolution filter (or kernel) that is containing the trainable weights of the neural network layer. In order to prevent the convolution operation from shrinking the feature map in width and height, the input layer can be padded with zeros. Image adapted from [47].

in the destination layer. In this case, max pooling is applied, which selects the maximum value of each pixel within a group to output at the destination layer. Other examples are min pooling, which selects the minimum value, or average pooling, which computes the average value.

Pooling is a form of downsampling and important in order to reduce computational complexity: as more convolutions are applied within a CNN, the feature maps gain more depth with pooling being used to shrink them with respect to their width and height. The reason that this has a small impact on the network's performance is that as more feature maps are computed, their spatial information becomes less important and their presence or absence represented by a lower amount of numerical values becomes more important. It is therefore important to choose a suitable pooling method in order to retain important information about the feature maps.

CNNs for Image Classification

Figure 2-18 shows a typical CNN architecture to perform **image classification**. Here, the input is a 3-channel image. The first four displayed steps show a repeated use of convolution, application of activation functions (here ReLU), and pooling. The further the information propagates through the network, the more depth the feature maps obtain and the less width and height they have due to pooling. This is the portion of the network that has learned to extract features that are ultimately useful in determining the class label of the image. Ideally, the CNN will have extracted features relevant for classification well enough such that only a

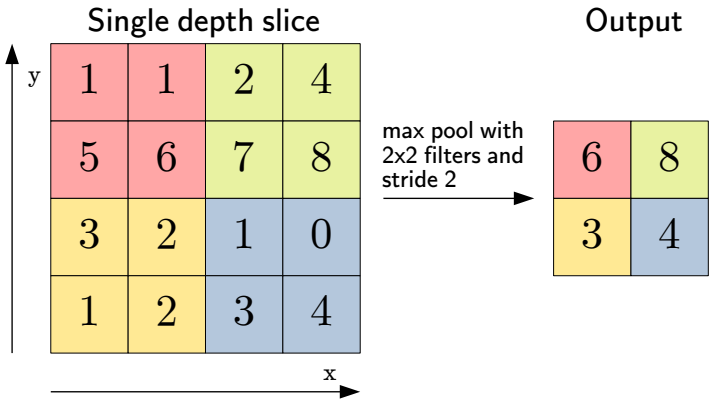


Figure 2-17: Example of max-pooling to reduce width and height of a feature map. Here, a filter of size 2×2 with stride 2 is used, meaning that the input map is separated into 2×2 windows in an interval (stride) of 2 pixels. The max operator then chooses the maximum value within each window to generate the next feature map. Image adapted from [46].

single fully connected layer of neurons is sufficient to determine the correct class label for the image based on those extracted features.

A notable CNN architecture that is relevant for this thesis is the Residual Network (ResNet) [48] as it is used as the backbone for the self-supervised learning algorithm Simple Framework for Contrastive Learning of Visual Representations (SimCLR) that is presented in Section 2-4. ResNet introduces the concept of *residual blocks*, which allow information to skip over one or several layers, effectively enabling the network to learn feature mappings that represent a trade-off between transformations via convolution and direct passthrough of information (please refer to Appendix D for a slightly more detailed explanation of ResNets and the architectures that it is built upon).

ResNets have been largely responsible for enabling the training of very deep CNNs. The depth of a ResNet architecture is specified by adding the amount of convolutional layers and the amount of skip connections. For example, ResNet-50 has 34 convolutional layers and 16 skip connections. The numbers appended to the ResNet name, such as 18, 34, 50, 101 or 152, indicate the total number of layers in the network, including both convolutional layers and skip connections.

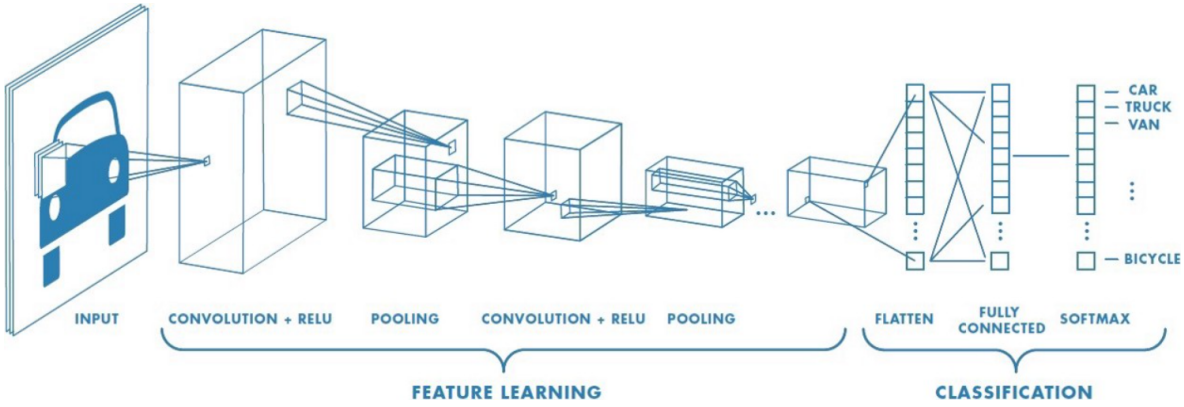


Figure 2-18: Diagram of a CNN for image classification. The 3-channel input image of a car is passed through several layers performing convolution and pooling operations, here referred to as the *feature learning* step. The result is a feature map that contains a, hopefully useful, feature representation of the input image. This feature representation serves as input to several fully connected layers, here titles as the *classification* step. For the final layer, the softmax function is chosen, which will cause the output vector to contain a list of probabilities that specify how likely it is that the input image belongs to one of the known class labels. Image from [46].

2-4 Self-supervised Learning for Vision Tasks

In the previous section, we have seen a simple diagram of a convolutional neural network (CNN) for image classification in Figure 2-18. The two major steps that are distinguished in this diagram are ***feature learning*** and ***classification***. When training such a network using supervised learning, the *feature learning* and *classification* steps happen at the same time: the prediction made by the fully-connected layers at the end of the network (the *classification* part), is evaluated using a loss function, whose gradient is propagated backwards through the network all the way to the very first input layer in the *feature learning* part. However, these two steps do not necessarily have to occur at the same time during training. Performing the feature learning step separately is a **key aspect of self-supervised learning approaches**, which aim to learn meaningful representations of the data without relying on explicit labels.

This section aims to provide an understanding of self-supervised learning approaches and their potential applications in various computer vision tasks. Section 2-4-1 will first discuss the intuition behind self-supervised learning and highlight the differences between contrastive and non-contrastive learning approaches. Section 2-4-2 will provide a detailed overview of the Simple Framework for Contrastive Learning of Visual Representations (SimCLR) — the self-supervised learning algorithm that was used to train the models to learn feature representations of tumor tissue images as part of this thesis work. Lastly, Section 2-4-3 will explain the linear evaluation protocol — a common method to evaluate the quality of feature representations that a self-supervised learning algorithm learns.

2-4-1 The Intuition Behind Self-supervised Learning

Consider the example of an image classifier separating images of cats and dogs: in supervised learning, we would train such a classifier on a large dataset containing images of cats and

dogs, which are labeled appropriately. However, this is very different from how humans learn to recognize objects [49]. Humans, especially when we are young, learn to recognize objects by repeatedly being exposed to them, "not by being told the name of every object" [50]. We may become experts in identifying objects before we learn their names and, in fact, before we have even learned a language at all. It is then just a matter of being taught the words corresponding to objects after which we can point to previously unseen objects, such as cats and dogs, and exclaim their names with high accuracy.

Albeit very simplified, this example highlights the intuition behind self-supervised learning: first, a model learns to extract meaningful features from the data in an unsupervised manner without explicit labels (**pre-training**), then it uses the learned features to perform a supervised task with a limited amount of labeled data (**fine-tuning** or **transfer learning**). This two-stage process allows the model to improve label efficiency, as it can leverage the structure and patterns present in the data itself during the first stage, reducing the need for extensive labeled data during the second stage.

However, it is not immediately obvious how one should design an algorithm that achieves the desired learning process outlined in the first stage. A crucial aspect of self-supervised learning involves creating **pre-text tasks**, which are carefully designed tasks that force the model to learn meaningful features from the data in order to solve them.

Pre-training: Contrastive Learning vs. Non-contrastive Learning

Self-supervised learning can be broadly categorized into **contrastive** and **non-contrastive** approaches [13]. The main difference between these approaches lies in the way training examples are generated and which type of pre-text task should be solved during **pre-training**.

Contrastive methods often involve designing a pre-text task that encourages models to learn representations that bring similar data points closer together (positive pairs) while pushing dissimilar data points further apart (negative pairs). One popular way to achieve this is by creating two views of the same image, such as adding color or cropping, and mapping them to similar representations (see Figure 2-20). State-of-the-art contrastive learning methods for vision tasks include MoCo [51, 52] and **SimCLR** (see Section 2-4-2) [14, 53].

Non-contrastive learning methods, on the other hand, provide only positive pairs and focus on solving a task related to the input data without explicitly modeling the relationships between different examples. In natural language processing, an example of this is to remove a word in the input text and define the pre-text task to determine the removed word based on the surrounding words. This is called context prediction, and an example in the visual domain is shown in Figure 2-19 [54]: Here, the pre-text task is to predict the relative locations of two randomly extracted patches from an image. State-of-the-art non-contrastive learning algorithms for vision tasks include BYOL [55], SwAV [56], and SEER [57].

After Pre-training: Fine-tuning vs. Transfer Learning and Feature Extraction

After the pre-training stage in self-supervised learning, models have acquired meaningful representations of the data. However, they need to be adapted for specific **downstream tasks**. This can be achieved through **fine-tuning** or **transfer learning and feature extraction**.

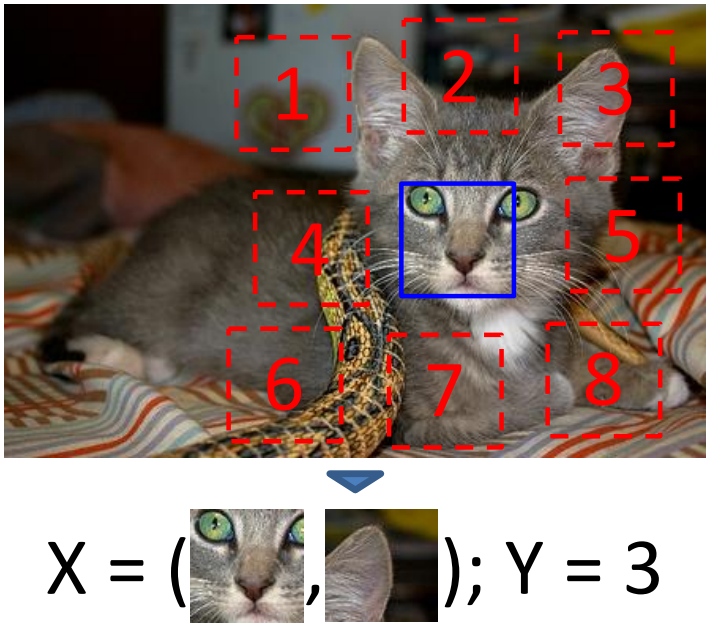


Figure 2-19: Example context prediction task for computer vision. The center crop (blue) is paired with a random surrounding crop (red) and given as inputs X to a network during pre-training. By learning to correctly predict the location Y of the image crops, the network learns about the spatial make-up of objects and the relationship between objects within images. The resulting feature representations the network learns are useful for computer vision tasks such as object detection. Image from [54].

Fine-tuning involves updating the entire pre-trained model with labeled data from the target task. In other words, the pre-trained model's layers are "unfrozen" and can be changed during the fine-tuning process.

Transfer learning, on the other hand, involves using the pre-trained model as a fixed **feature extractor**. In this approach, the pre-trained model is used to extract features from the labeled data of the target task, and these features are then fed into a new model, e.g., a classifier, which is trained from scratch. This process leaves the pre-trained model unchanged ("frozen"), and only the new model is updated during training. The linear evaluation protocol explained in Section 2-4-3 is an application of transfer learning.

2-4-2 SimCLR: A Simple Framework for Contrastive Learning of Visual Representations

SimCLR (Simple Framework for Contrastive Learning of Visual Representations), proposed by Chen et al., is a state-of-the-art contrastive learning algorithm that focuses on learning feature representations from unlabeled data by recognizing similarities between differently augmented views of the same base image and recognizing dissimilarities between augmented views of different base images [14, 53].

The pre-training stage of SimCLR is illustrated in Figure 2-20 and can be summarized in the following steps:

1. Randomly augment an input image twice, generating two distinct views of the same base image. Common augmentations include flipping, rotating, cropping, resizing, etc. as shown in see Figure 2-21.
2. Independently feed both augmented views of the image into an encoder CNN to obtain intermediate feature representations (displayed in blue).
3. Process the intermediate feature representations using a MLP that serves as a projection head, resulting in feature projections (displayed in violet).
4. Optimize the model using a contrastive loss function that maximizes the similarity between projections of the two augmented views of the same base image (positive pairs, green "Attract" arrows), while minimizing the similarity between projections of different base images (negative pairs, red "Repel" arrows).
5. Once the pre-training is complete, use the intermediate feature representations output by the encoder CNN as input to other networks for downstream tasks.

Note that all the networks denoted with CNN (convolutional neural network) and MLP (multilayer perceptron), respectively, share the same weights during training. In the standard implementation of SimCLR, the CNN is a ResNet encoder and the MLP is a fully-connected network with 3 hidden layers and ReLU (Rectified Linear Unit) activations.

The SimCLR Algorithm and the NT-Xent Loss Function

Figure 2-22 shows a symbolic representation of the diagram shown in Figure 2-20. Here, $\boldsymbol{x}$ represents an input image, which is augmented by transformations t and t', resulting in the augmented image pair $\tilde{\boldsymbol{x}}_i$, $\tilde{\boldsymbol{x}}_j$. The transformations are randomly drawn from the set of all pre-defined transformations $\mathcal{T}$. A base encoder $f(\cdot)$ (represented by a CNN in Figure 2-20) processes the augmented image pair and computes their **feature representations** $\boldsymbol{h}_i$, $\boldsymbol{h}_j$. The feature representations are further processed by a projection head $g(\cdot)$ (represented by a MLP in Figure 2-20), which computes their **feature projections** $\boldsymbol{z}_i$, $\boldsymbol{z}_j$.

As a pre-training loss function SimCLR uses a variation of the InfoNCE (Information Noise-Contrastive Estimation) loss [59], which the authors call NT-Xent (normalized temperature cross-entropy) loss

$$\mathcal{L}^{\text{NT-Xent}} = -\frac{1}{N} \sum_{i,j \in \mathcal{MB}} \log \left(\frac{\exp\left(\text{sim}\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right)/\tau\right)}{\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i]} \exp\left(\text{sim}\left(\boldsymbol{z}_i, \boldsymbol{z}_k\right)/\tau\right)} \right). \tag{2-32}$$

Here, N is the amount of images (before generating augmentation pairs), $i, j \in \mathcal{MB}$ corresponds to all positive pairs in the minibatch (after generating augmentation pairs), $\boldsymbol{z}_i$ is the feature projection of the ith image in the minibatch, τ is a temperature normalization constant, and $\mathbf{1}_{[k \neq i]}$ is an indicator function that is equal to 1 iff $k \neq i$ and 0 otherwise. $\text{sim}(\mathbf{a}, \mathbf{b})$ is the cosine similarity between two vectors $\mathbf{a}$ and $\mathbf{b}$:

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} \tag{2-33}$$

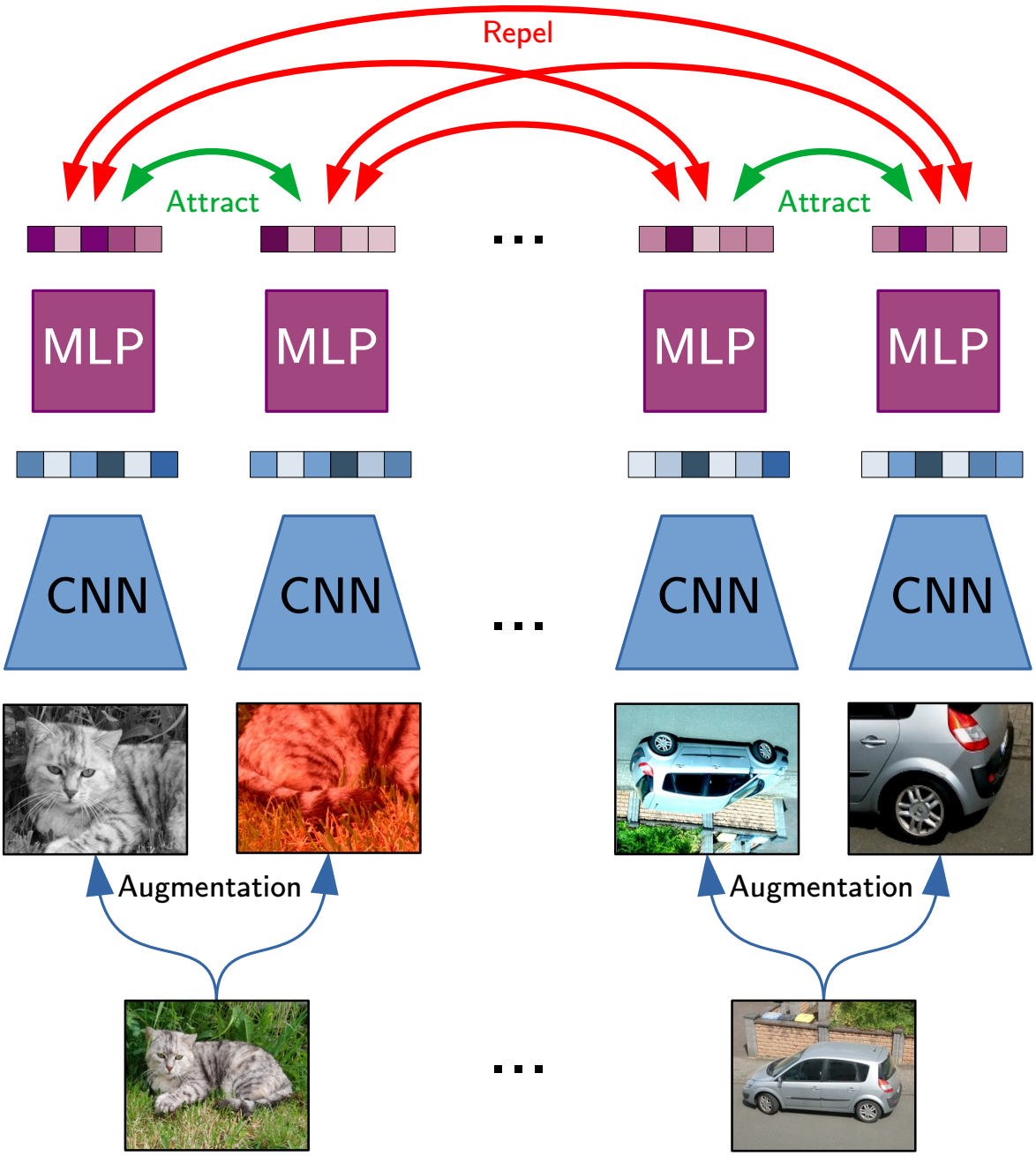


Figure 2-20: Diagram of the SimCLR pre-training procedure. The image of the cat and car are augmented in two different ways and fed into the encoder CNN. The resulting intermediate feature representations are further processed by a MLP whose outputs serve as the training signal: the two representations originating from the cat image and car image, respectively, should be as similar as possible. The representations originating from another image should be as dissimilar as possible. The intermediate feature representation as output by the encoder CNN can then serve as input to other networks for downstream tasks. Note that all the networks denoted with CNN and MLP, respectively, share the same weights during training. Image adapted from [58].

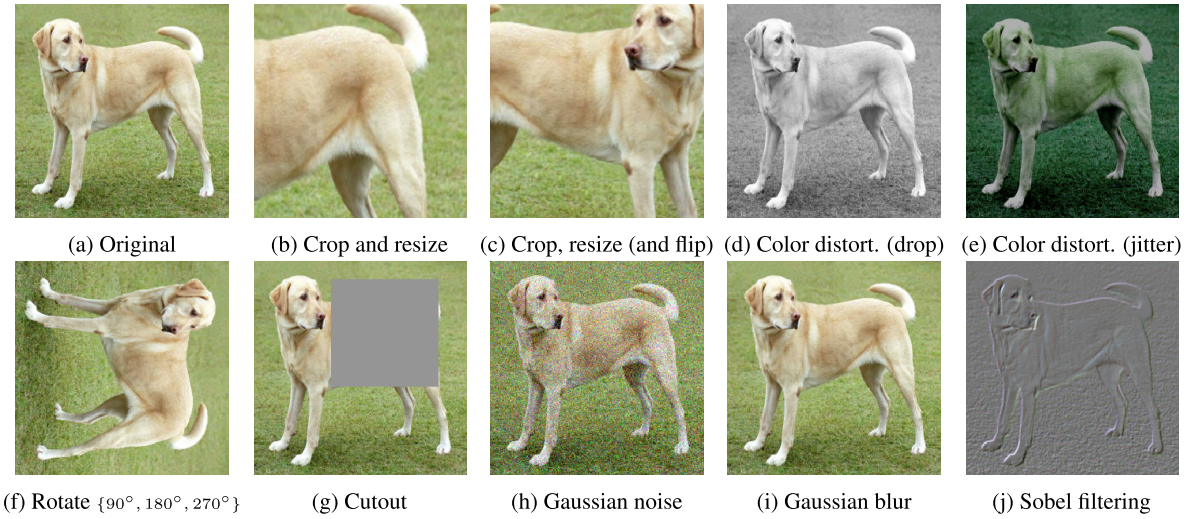


Figure 2-21: Example of typical image augmentations for deep learning. Augmentations are generally applied to image datasets in order to artificially increase sample size as well as making the trained network more robust against noise and other image variations. In SimCLR specifically, these augmentations are applied twice to the same image and the pre-training task is to be able to recognize when an image pair originated from the same source image or not. Image from [14].

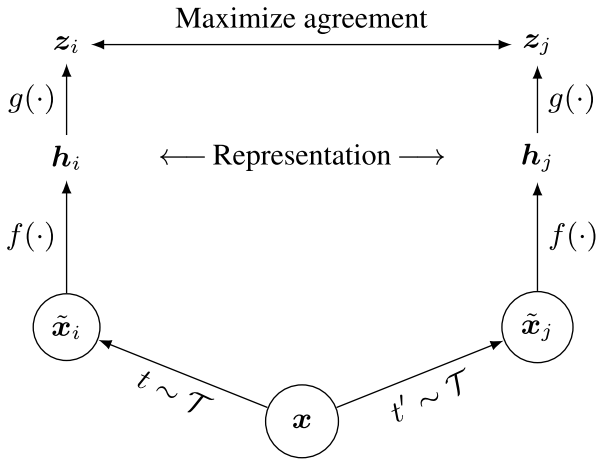


Figure 2-22: Symbolic diagram of the SimCLR pre-training procedure. The input image $\boldsymbol{x}$ is augmented two separate ways by transformations randomly drawn from $\mathcal{T}$, resulting in an augmented image pair $\boldsymbol{x}_i$, $\boldsymbol{x}_j$. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement between feature projections $\boldsymbol{z}_i$, $\boldsymbol{z}_j$. After training, $g(\cdot)$ is discarded and only $f(\cdot)$ is used to compute representations $\boldsymbol{h}$ for downstream tasks [14].

Figure 2-23: Three example images of handwritten digits. The "4", "7", and "9" digits all share similar features (e.g., diagonal lines, horizontal lines near the top of the digit). In the context of SimCLR, a lower value of τ in the NT-Xent loss function may be necessary to better distinguish between these very similar images. Digit images are taken from the MNIST dataset [61].

By making use of logarithmic properties, the function can be re-arranged as [60]

$$\mathcal{L}^{\text{NT-Xent}} = \underbrace{-\frac{1}{N}\sum_{i,j\in\mathcal{MB}} \text{sim}\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right)/\tau}_{=\mathcal{L}_\text{A}} \underbrace{+\frac{1}{N}\sum_{i}^{N}\log\left(\sum_{k=1}^{2N}\mathbf{1}_{[k\neq i]}\exp\left(\text{sim}\left(\boldsymbol{z}_i, \boldsymbol{z}_k\right)/\tau\right)\right)}_{=\mathcal{L}_\text{D}} \tag{2-34}$$

where the **alignment loss** $\mathcal{L}_\text{A}$ and the **distribution loss** $\mathcal{L}_\text{D}$ correspond to the numerator and denominator in Eq. (2-32), respectively. $\mathcal{L}_\text{A}$ can be considered the non-contrastive part of the loss function as it only evaluates **similarity of positive pairs**, whereas $\mathcal{L}_\text{D}$ is the contrastive part which evaluates **similarity of negative pairs**.

The Effect of the Temperature Parameter τ

The temperature parameter τ in the NT-Xent loss function plays an important role in controlling the scale of the similarity scores between feature projections. By dividing the cosine similarity by τ, the similarity scores are normalized, affecting the sharpness of the distribution of these scores. In other words, the temperature parameter influences the relative importance of positive and negative pairs during optimization.

Consider the example of three handwritten digits in Figure 2-23 where the "4","7", and "9" digits all share similar features (e.g., diagonal lines, horizontal lines near the top of the digit). If we set a large value of τ in the NT-Xent loss function during training, the similarity scores between the feature projections of these similar images will be normalized to similar values. As a result, the model might treat these similar images as equally similar, potentially causing confusion during pre-training and resulting in worse feature representations.

On the other hand, if we set a smaller value of τ, the similarity scores will be more sensitive to the differences between these similar images, leading to a sharper distribution of scores. This means that the model will be better able to distinguish between these similar images during pre-training and learn better feature representations as a result.

In certain situations, however, a larger value of τ can be advantageous. For example, consider a dataset with a high degree of intra-class variation, where instances of the same class exhibit

a wide range of features or characteristics (e.g., many different cat and dog breeds in a cat and dog image dataset). In such cases, using a larger value of τ can help the model focus on the broader similarities between instances of the same class, rather than being overly sensitive to small differences.

Augmentations in SimCLR

The role of data augmentations in SimCLR is crucial, as they directly influence the quality of the learned representations. Chen et al. [14] observed that the choice of augmentations significantly affects the performance of SimCLR. In particular, they found that using multiple augmentations, such as cropping with random resizing, flipping, color jittering, and blurring, was more effective than using a single augmentation. This is because a diverse set of augmentations allows the model to learn more robust and generalizable features that are useful for a wide range of downstream tasks.

Tian et al. [19] have investigated the effect of augmentations on contrastive learning performance and argued that a good set of augmentations should reduce mutual information between views while keeping task-relevant information intact. In other words, the augmentations should introduce sufficient variation between the views of the same image to challenge the model, but not to the extent that the underlying semantic content is altered or lost. This balance enables the model to focus on learning features that are relevant to the task at hand and less sensitive to irrelevant variations introduced by the augmentations.

Jing et al. [62] have found that augmentations that are too strong will result in dimensional collapse of the feature representations. This means that, when the augmentations are overly aggressive, the learned feature representations may become overly simplified, losing their discriminative power and collapsing into a low-dimensional subspace. This limits the model's ability to distinguish between different images and may lead to poor performance on downstream tasks. Therefore, it is important to carefully select and tune the augmentations to achieve the optimal trade-off between encouraging the model to learn robust features and preserving the discriminative power of the learned representations.

2-4-3 Assessing Pre-trained Encoders with Linear Evaluation

The goal of pre-training an encoder network using self-supervised learning is to learn high-quality **feature representations** it generates from the given input. There are multiple methods to evaluate the quality of these feature representations, but here we will only highlight the most popular [13] one: **linear evaluation**, which was introduced by Zhang et al. [63, 64] and involves training a linear classifier on top of the learned feature representations.

Figure 2-24 illustrates the process: the pre-trained encoder's output feature representations are used as input for the supervised training of a linear classifier. The performance of the classifier reflects the quality of the feature representations learned by the pre-trained encoder. Since a linear classifier has low discriminative power (i.e., it has a low capacity), its ability to predict the correct class is highly dependent on the quality of the feature representations. Consequently, the performance of the trained linear classifier serves as an indirect measure of the quality of the pre-trained encoder's feature representations.

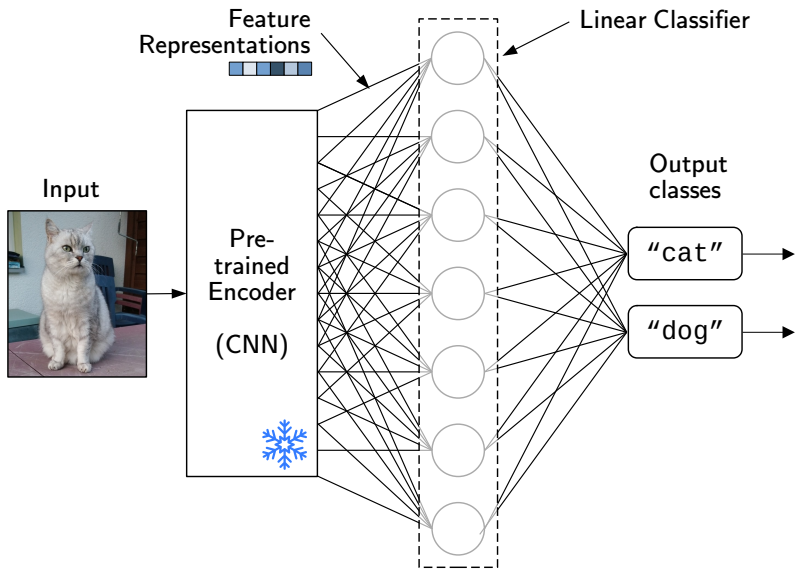


Figure 2-24: Schematic overview of linear evaluation of a pre-trained encoder. The pre-trained encoder computes a feature representation of the given input image. This feature representation serves as input to a linear classifier, which consists of a single layer of neurons. The linear classifier is trained to predict the correct class with supervision (i.e., labels are available). The pre-trained encoder's weights remain unchanged (frozen) during this.

During linear evaluation, the weights of the pre-trained encoder remain unchanged (frozen) while training the linear classifier. This approach allows for an assessment of the feature representations' quality without modifying the pre-trained model. The performance of each linear classifier, specifically trained for its corresponding pre-trained encoder, can then be used as a benchmark for comparing the effectiveness of different pre-trained encoders in learning meaningful feature representations.

2-5 Related Work

In recent years, significant progress has been made in the field of tumor analysis, with the development of various computational tools and approaches aimed at better understanding tumor tissues and the tumor microenvironment (TME). This section provides an overview of the current state of the art tools, as well as techniques that leverage self-supervised learning — specifically SimCLR. Lastly, a discussion of the limitations of these existing tools and techniques is provided, and how the work in this thesis can address them.

2-5-1 State of the Art Tools for Tumor Analysis

QuPath, short for Quantitative Pathology, is an open-source tool for digital pathology and whole slide image analysis [65]. It can be used to perform various image analysis tasks, including cell detection, segmentation, and classification. In the context of tumor tissue analysis, QuPath can be used to classify individual cells based on their morphological and phenotypic features. The software allows users to train and apply ML algorithms, such as random forests and support vector machines, to differentiate between various cell types, including tumor cells, immune cells, and stromal cells. Additionally, QuPath offers the possibility to analyze spatial relationships between different cell populations, providing a better understanding of the TME.

CellProfiler is an open-source image processing and analysis software that enables high-throughput analysis of cell-based assays [66]. It can be used for various tasks, notably for segmentation and quantification of individual cells in images. In the context of tumor tissue analysis, CellProfiler can be employed to preprocess histological images, segment cells, and extract relevant features for classification. CellProfiler's flexible pipeline system allows users to integrate various image processing modules to tailor the analysis workflow to their specific needs. It is often complemented by **ImageJ** [67], an open-source tool that focuses on single-image processing and investigation. Both CellProfiler and ImageJ have established themselves as leading open-source platforms in the field of bioimage analysis with thousands of citations [68].

TME-Analyzer (developed at Erasmus University Medical Center) is a versatile tool designed to facilitate the analysis of TMEs and address the challenges posed by high patient-to-patient and tissue-to-tissue variations in MxIF image intensities [7]. The TME-Analyzer workflow is organized into several key stages: image loading, foreground selection, tissue segmentation, nucleus/cell segmentation, cell phenotyping, and data analysis and exportation. By offering the user an intensity profile histogram during the foreground selection and tissue segmentation steps, the tool allows for easy threshold selection and correction for uneven background signals. Nucleus segmentation can be performed manually using a watershed algorithm [69] or by using a pre-trained neural network called StarDist [70]. By applying Voronoi segmentation to the segmented nuclei, cell segmentation masks can be generated. TME-Analyzer has demonstrated its utility in the analysis of triple-negative breast cancer (TNBC) tissue samples, where it was used to obtain a classifier that significantly predicted short versus long survivors and was benchmarked against inForm [71, 72], a commercially available tool for MxIF image analysis.

2-5-2 Applications of SimCLR for Tumor Analysis

Self-supervised learning has emerged as a promising approach to address the scarcity of labeled datasets in medical image analysis. Here, we will look at two studies related to the problem of tumor analysis using the self-supervised learning algorithm SimCLR (Simple Framework for Contrastive Learning of Visual Representations).

Self-supervised contrastive learning for digital histopathology by Ciga et al.

In their study, Ciga et al. [16] utilized the SimCLR algorithm to analyze digital histopathology data. A diagram of the pre-training pipeline is shown in Figure 2-25 (note the similarity to Figure 2-20). The authors investigated the use of several pre-training augmentations, including random flipping, rotations, Gaussian blurring, and color jittering.

Ciga et al. pre-trained the SimCLR models on 57 histopathology datasets without any labels and found that combining multiple multi-organ datasets with different staining and resolution properties enhanced the quality of the learned features. They also found that using more images for pre-training led to better performance in the downstream tasks of tissue classification and segmentation. They highlighted that the success of the contrastive pre-training heavily relies on the diversity of the unlabeled training set, rather than the number of images. Furthermore, they noted that the organ from which the training images were extracted did not substantially affect the quality of learned representations. This is a significant insight, since it suggests that SimCLR models can improve by learning from more datasets than just those that focus on a specific organ and implies that a single model may successfully learn features for a broad range of tissue types.

The study focused on pre-training networks to learn features from images on a tissue-level rather than individual cells. The authors point out that images that are visually similar but only feature small variations, for example single-cell tumors in a larger tissue image, "were not suitable for contrastive learning, and led to noisy representations" [16], and that addressing those domain-specific issues requires further research.

NaroNet: Discovery of tumor microenvironment elements from highly multiplexed images

NaroNet is a deep learning framework that can analyze TMEs and provide patient-level predictions [18]. The NaroNet pipeline analyzes tumors in three levels of spatial complexity thanks to its multi-stage layout:

- The patch-contrastive learning (PCL) module employs the SimCLR algorithm and learns features from small image patches extracted from the input images,
- the features learned by the PCL are arranged in a graph that captures the spatial relationship of the original image patches,
- the graph is fed into a series of graph neural networks, which learn higher-level features ("neighborhood learning" and "area learning").

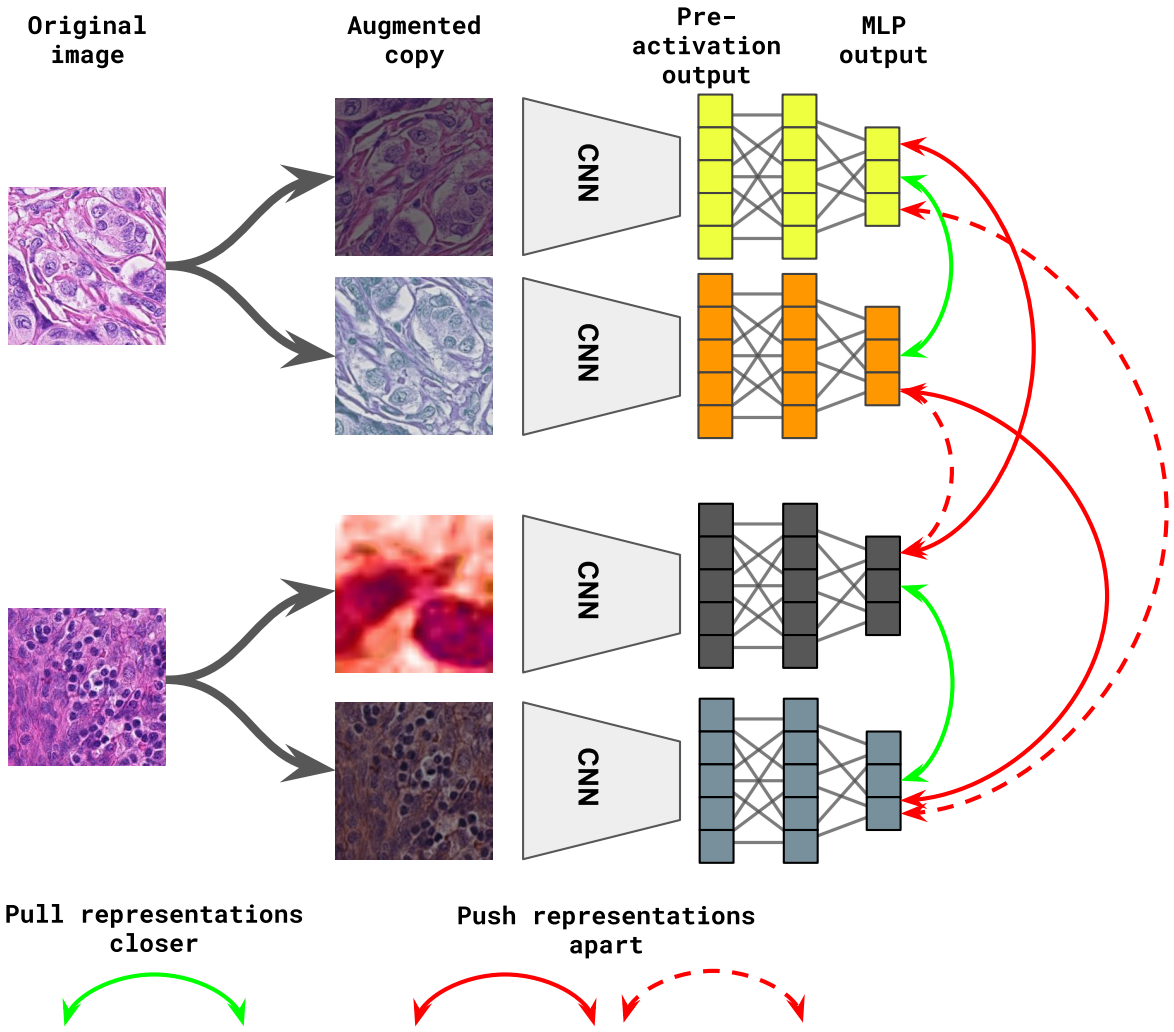


Figure 2-25: Diagram of the SimCLR pre-training procedure applied by Ciga et al. Note the similarities to Figure 2-20. The networks were pre-trained on various datasets featuring brightfield microscopy images of a broad range of tissue types. Input images were of size 224×224 pixels and captured larger tissue areas rather than individual cells. Image from [16].

A diagram of the pre-training pipeline for the PCL is shown in Figure 2-26 (note the similarity to Figure 2-20). The PCL learns from small image patches of sizes between 10×10 and 15×15 pixels, which are randomly extracted from the input MxIF images with the condition that a patch must contain one or more cells. The only augmentations used during pre-training are random rotations, flipping and cropping of the extracted patches.

By using the features learned by the PCL, whole tissue images can be represented in a compressed yet enriched format, which NaroNet utilizes in its downstream graph neural networks to perform TME analysis.

2-5-3 Limitations of Previous Work

Although tools for bioimage analysis such as QuPath, CellProfiler, TME-Analyzer, etc. are very capable, their main limitation stems from the fact that they are, at best, semi-automatic. The deep learning capabilities that these tools offer are either facilitated by networks that were (supervisedly) trained on separate data or require the users to provide labels on their own if they want to train models on their data. Even unsupervised learning capabilities require tuning of hyperparameters and the obtained results can be difficult to interpret.

Since these traditional tools require expert users to perform annotation, labeled datasets in bioimaging are scarce as a result. Self-supervised learning techniques is a way to address this label scarcity and the work by Ciga et al. [16] as well as NaroNet [18] show that useful features can be learned from tissue images. Furthermore, these features can be used to improve the performance of downstream applications.

While the results obtained by Ciga et al. are promising, it is not clear if their proposed networks would work well on MxIF images, since their study only considered brightfield microscopy images. The difference in image modality may require an entirely different consideration for pre-training augmentations. Furthermore, MxIF images often have more than 3 color channels, which means that typical neural network architectures may not be suitable as those are primarily focused on analysis of natural images.

NaroNet is capable of analyzing MxIF images, but its PCL architecture is limited to analyzing datasets of images that all have the same amount of image channels. Given the results by Ciga et al., a big advantage of self-supervised learning on tissue images is that data from various organs and tissue types can improve performance. Since not all MxIF image datasets have the same amount of image channels (or even target the same immune cells), NaroNet is limited to analyzing images from small subsets of all available data.

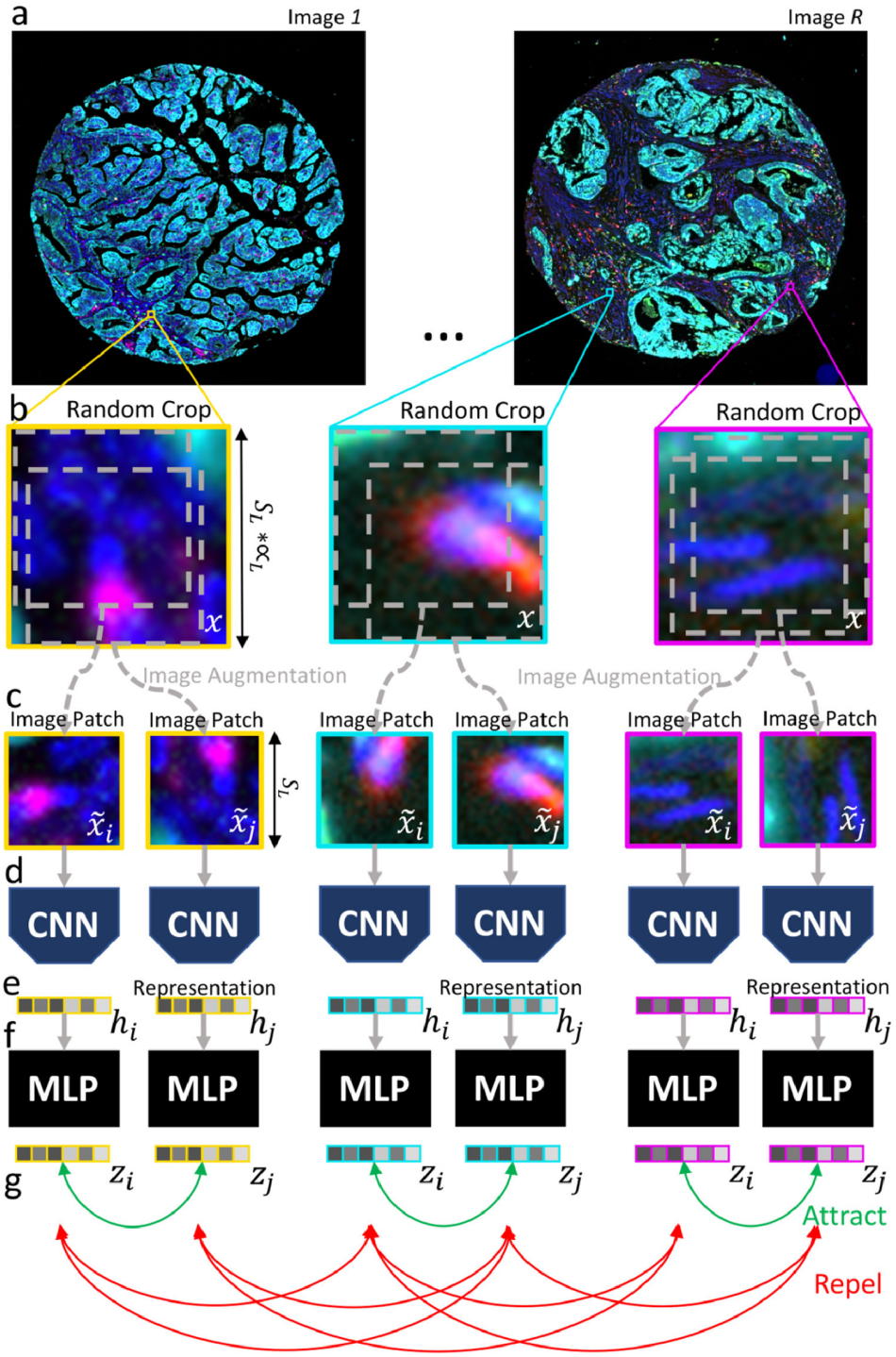


Figure 2-26: Diagram of the SimCLR pre-training procedure of NaroNet's PCL module. Note the similarities to Figure 2-20. The PCL is trained on small image patches (sizes between 10×10 and 15×15 pixels) randomly extracted from the tissue images, with the condition that one or more cell must be present in an extracted patch. Image from [18].

Chapter 3

Methodology

This chapter describes the steps taken to perform the training and evaluation of the self-supervised neural networks.

Section 3-1 gives an overview of the algorithms and evaluation metrics used. Section 3-2 presents the main dataset used for the training — the triple-negative breast cancer (TNBC) dataset — and details every step taken to adjust the dataset to be used for the purpose of tumor section classification and individual cell classification. Section 3-3 presents two more datasets that were used — the CellPose and STL-10 datasets. Lastly, Section 3-4 and Section 3-5 detail the steps taken in the two training experiments: tissue section classification and individual cell classification, respectively.

3-1 Algorithms and Evaluation Metrics

The training procedure for a complete network, capable of being evaluated, can be summarized in two steps following the outline of self-supervised learning algorithms in Section 2-4:

1. Self-supervised pre-training of an encoder network according to the SimCLR algorithm (Simple Framework for Contrastive Learning of Visual Representations) [14, 53].
2. Supervised training of a linear classifier on top of the pre-trained encoder.

To evaluate the quality of the feature representations learned by the pre-trained encoders following the linear evaluation protocol (Section 2-4-3), I trained the linear classifiers on top of the base encoders to perform two separate classification tasks, along with their respective evaluation metrics as follows:

1. Determine the inflammation status of entire tumor sections in TNBC images (described in Section 3-4).
   Evaluation metric: Classification accuracy (see Section 2-2-3).

Table 3-1: Summary of spatial immunophenotypes per patient, and corresponding images in the TNBC dataset. The spatial immunophenotype labels are assigned per patient and therefore every image belonging to a particular patient inherits this label.

Spatial immuno-phenotype	Label	Patients	Images
Excluded	`"excl"`	17 (27.4%)	272 (26.9%)
Ignored	`"ign"`	24 (38.7%)	389 (38.5%)
Inflamed	`"infl"`	21 (33.9%)	349 (34.6%)
Total		62	1010

2. Determine the phenotypes of individual immune cells within tumor sections in TNBC images (described in Section 3-5).
   Evaluation metric: The classifier's Area Under the Precision-Recall Curve (PR-AUC) (see Section 2-2-3).

It is important to note that the linear classifiers serve a dual purpose in this context: they not only perform tasks of interest but also provide a means to evaluate the quality of the base encoder's feature representations through their performance on these tasks.

3-2 TNBC Dataset

The TNBC (triple-negative breast cancer) dataset is the primary dataset used in this thesis for training the networks and evaluating their performance. It consists of 1010 multiplex immunofluorescence (MxIF) microscopy images of TNBC tissue sections from 62 patients. These images have been gathered as part of a previous study at Erasmus University Medical Center (Erasmus MC) [4] and have been labeled using TME-Analyzer [7]. Notably, the available labels include the identified spatial immunophenotype **per patient** (Table 3-1) as well as the location and cell phenotype of **every located cell** within the images (Table 3-2).

Pre-processing of the Image Data

The original images are available as size $1340 \times 1004 \times 8$ (width×height×color channels) `.tif` files with 32-bit color depth. As a first pre-processing step, I split the 8 individual channels into separate `.png` files with 8-bit color depth and normalized their intensities such that the dimmest pixel obtained intensity 0 and the brightest pixel obtained intensity 255. These images were used for training the networks for tissue image classification.

For individual cell classification, I used the cell location information as determined by TME-Analyzer and extracted small image patches of size 64×64 pixels centered around every cell.

Table 3-2: Summary of individual cell phenotypes in all images in the TNBC dataset. Note that a single cell can be assigned multiple phenotype labels (multilabel): the given "Cells" quantities specify the amount of cells that were assigned a particular label. Cells that were assigned none of the labels of interest were assigned the `"other"` label. The last row specifies the total amount of cells that were identified by TME-Analyzer.

Cell phenotype	Label	Cells
Cytokeratin	`"CK+"`	726 101 (31.64%)
CD3	`"CD3+"`	140 426 (6.12%)
CD8	`"CD8+"`	190 615 (8.31%)
CD20	`"CD20+"`	108 444 (4.73%)
CD56	`"CD56+"`	5 291 (0.23%)
CD68	`"CD68+"`	54 700 (2.38%)
Other cells	`"other"`	1 226 886 (53.46%)
Total		2 295 083

Train/test Split

For all experiments involving supervised learning, unless stated otherwise, I performed an identical split of the dataset **on a patient level** putting 47 (75.8 %) patients into the training set and 15 (24.2 %) patients into the test set. Subsequently, this results in a split of 761/249 (75.3 %/24.7 %) on an image level and 1 732 096/562 987 (75.5 %/24.5 %) on an individual cell level.

The following subsections describe how the dataset was further processed for the purposes of tumor section classification (Section 3-2-1) and individual cell classification (Section 3-2-2).

3-2-1 TNBC Dataset for Tumor Section Classification

For the task of entire tumor section classification, I applied the following change to the patient-level spatial phenotype label data:

- Combine labels `"excl"` ("excluded" phenotype) and `"ign"` ("ignored" phenotype) into the new label `"non-infl"` ("non-inflamed" phenotype).

The resulting training and test sets are summarized in Table 3-3.

To give a visual impression of the image data that is being analyzed when classifying tumor sections, Figure 3-1 shows an example tissue image from the TNBC dataset, where Figure 3-1a is a composite image of all color channels as it would be considered when analyzed by humans, and Figures 3-1b through 3-1i are the respective individual channels (note: for better visibility in this document, these images have their colors inverted, their brightness reduced by 20 %, and contrast increased by 40 %).

Table 3-3: Summary of training and test sets for the task of tumor section classification.

Spatial immuno-phenotype	Label	Images, training set	Images, test set
Non-inflamed	`"non-infl"`	497 (65.3%)	164 (65.9%)
Inflamed	`"infl"`	264 (34.7%)	85 (34.1%)
Total		761	249

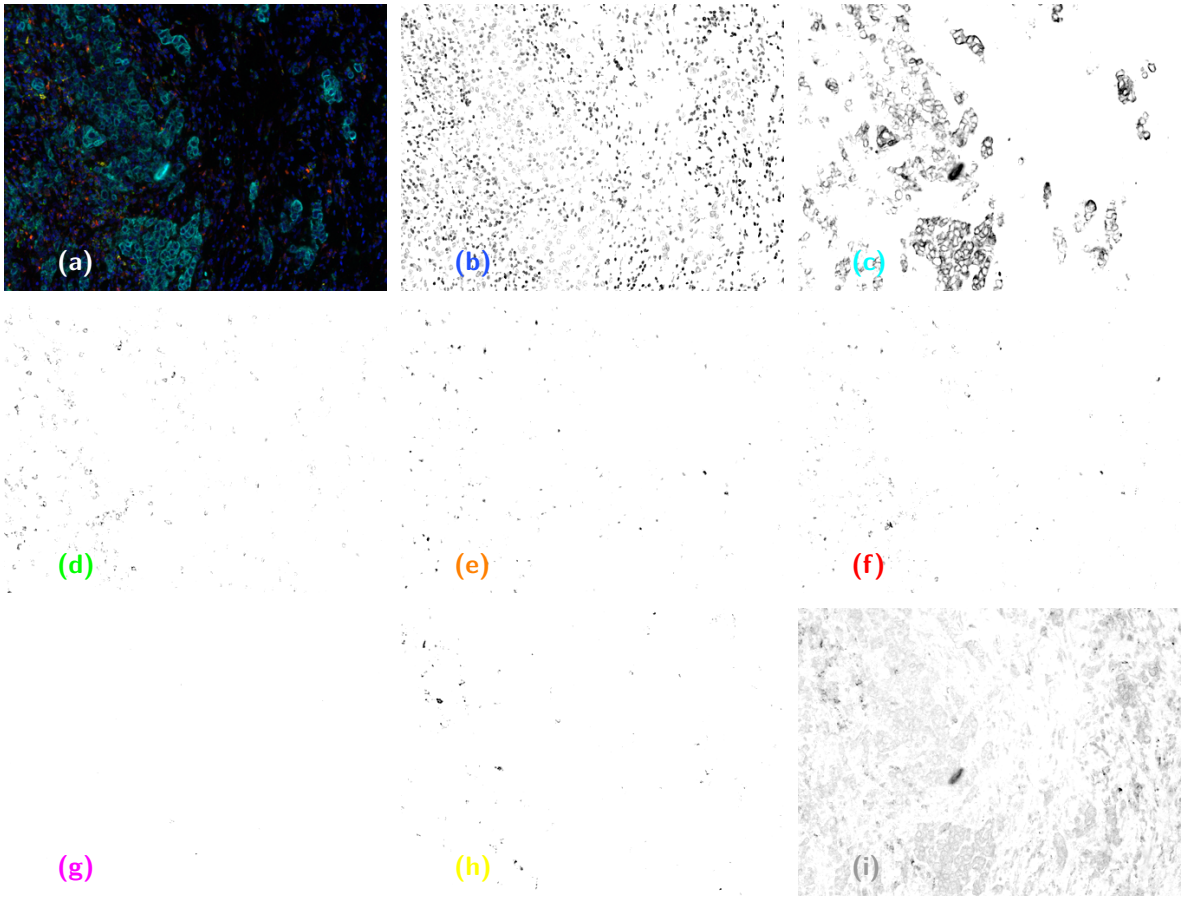


Figure 3-1: Example image from the TNBC dataset. **(a)** shows a composite of 7 color channels (excluding the background channel) flattened into a 3-channel RGB image. The remaining images show the intensity-normalized individual color channels in grayscale (for better visibility in this document, the colors have been inverted, brightness reduced by 20 %, and contrast increased by 40 %): **(b)** DAPI, **(c)** CK (Cytokeratin), **(d)** CD3, **(e)** CD68, **(f)** CD8, **(g)** CD56, **(h)** CD20, **(i)** background. Note that the colors of the subfigure indices from **(b)**–**(h)** correspond to the colors these respective channels contribute to the composite in **(a)**.

Table 3-4: Summary of the TNBC dataset for the task of individual cell classification with training set and test set split. The training set was balanced to mitigate class imbalance issues during supervised training. The imbalanced training set (displayed in gray) was not used during experiments. Note that a single cell can be assigned multiple phenotype labels (multilabel), so percentage values do not add up to 100 %. Cells that were assigned none of the labels of interest were assigned the `"other"` label. The last row specifies the total amount of cells contained within a set.

Cell phenotype	Label	Image patches, training set imbalanced	Image patches, training set balanced	Image patches, test set
Cytokeratin	`"CK+"`	530 458 (30.63%)	4 698 (19.84%)	195 643 (34.75%)
CD3	`"CD3+"`	116 036 (6.70%)	4 984 (21.04%)	24 390 (4.33%)
CD8	`"CD8+"`	157 270 (9.08%)	4 760 (20.01%)	33 345 (5.92%)
CD20	`"CD20+"`	95 088 (5.49%)	5 235 (22.10%)	13 356 (2.37%)
CD56	`"CD56+"`	4 698 (0.27%)	4 698 (19.84%)	593 (0.11%)
CD68	`"CD68+"`	40 138 (2.32%)	4 913 (20.74%)	14 562 (2.59%)
Other cells	`"other"`	919 274 (53.07%)	4 698 (19.84%)	307 612 (54.64%)
Total		1 732 096	23 684	562 987

3-2-2 TNBC Dataset for Individual Cell Classification

For the task of individual cell classification, in order to mitigate the effects of class imbalance during supervised training, I balanced **only the training set** such that every cell phenotype is represented in approximately equal proportion. The resulting training and test sets are summarized in Table 3-4.

Since the `"CD56+"` label is the least common in the dataset, all other labels were undersampled to occur in similar quantities, resulting in every phenotype label being represented in about 20 % of cells in the training set (note that a single cell can be assigned multiple phenotypes, so these percentages will not add up to 100 %). For completeness, the training set before applying this balancing is also shown in Table 3-4, **but this imbalanced training set was not used during the experiments**.

To give a visual impression of the image data that is being analyzed when classifying individual cells, Figure 3-2 shows an example cell image patch from the TNBC dataset, where Figure 3-2a is a composite image of all color channels as it would be considered when analyzed by humans, and Figures 3-2b through 3-2i are the respective individual channels (note: for better visibility in this document, these images have their colors inverted).

3-3 Other Datasets

For the purpose of investigating the effect of including image data from outside the TNBC dataset during pre-training, I used two more publicly available datasets: CellPose [73] (Section 3-3-1) and STL-10 [74] (Section 3-3-2).

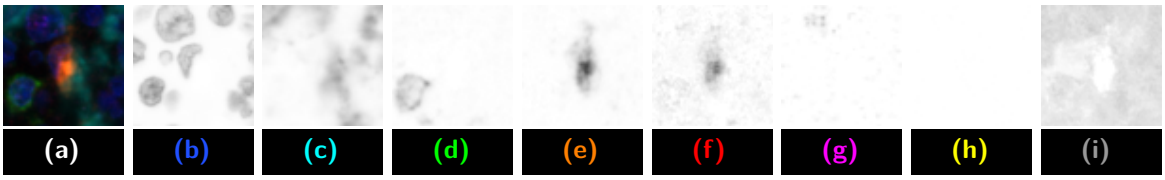


Figure 3-2: Example 64×64 image patch centered around a cell from the TNBC dataset. Similar to Figure 3-1, **(a)** shows a composite of 7 color channels (excluding the background channel) flattened into a 3-channel RGB image. The remaining images show the intensity-normalized individual color channels in grayscale (with inverted colors for better visibility): **(b)** DAPI, **(c)** CK (Cytokeratin), **(d)** CD3, **(e)** CD68, **(f)** CD8, **(g)** CD56, **(h)** CD20, **(i)** background. Note that the colors of the subfigure indices from **(b)**–**(h)** correspond to the colors that these respective channels contribute to the composite in **(a)**. We can confirm that the image patch is centered around a cell by looking at the DAPI channel, which shows that a cell nucleus is indeed present near the center of the image patch. However, assigning the cell to a phenotype is not trivial, since the CK, CD68, and CD8 channels have a high signal intensity near the image center. The chosen image patch size of 64×64 pixels results in surrounding cells being generously included in the patch — this will provide the encoder some spatial information that it can learn in order to produce better feature representations of the cell of interest.

3-3-1 Cellpose Dataset

The Cellpose dataset comprises a diverse set of images that encompass a wide range of cell types and other experimental conditions, which allows for the development and evaluation of robust image classification and segmentation algorithms [73]. The dataset was created with the goal of enabling improved performance in various cell analysis tasks, such as identifying cell types, understanding cell behavior, and tracking cell movement over time.

The images in the dataset are derived from both fluorescence microscopy and bright-field microscopy, as well as photographs of objects arranged in various patterns. Some example images are shown in Figure 3-3.

Pre-processing of the Image Data

The CellPose dataset consists of about 600 images with sizes around 512×512 pixels and between 1 and 3 color channels in `.png` format with a color depth of 8-bit. The dataset was only used during experiments for individual cell classification. In order to integrate its images into the training process, I split the original images into its individual color channels (if applicable) and randomly extracted a total of 102 400 image patches of size 64×64 pixels from the CellPose dataset.

The CellPose dataset was not used for supervised training and I therefore discarded its label data.

3-3-2 STL-10 Dataset

The STL-10 dataset is a popular benchmark dataset used in computer vision research for image classification tasks [74] and is based on the ImageNet dataset [75]. The images feature

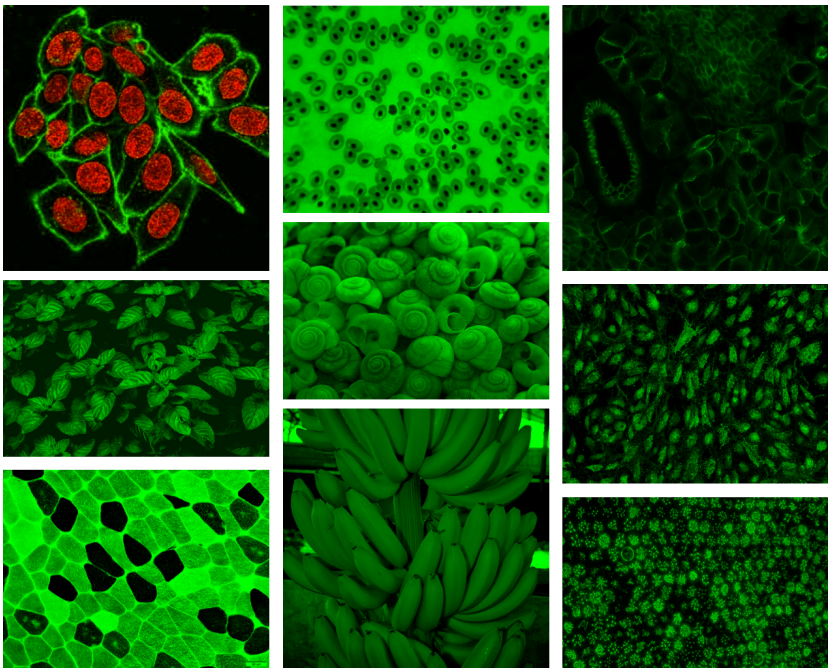


Figure 3-3: Example images from the Cellpose dataset. Aside from typical microscopy images from various modalities, the dataset includes unusual images of objects such as bananas or seashells arranged in patterns.

objects from 10 classes with some examples shown in Figure 3-4: `"airplane"`, `"bird"`, `"car"`, `"cat"`, `"deer"`, `"dog"`, `"horse"`, `"monkey"`, `"ship"`, and `"truck"`.

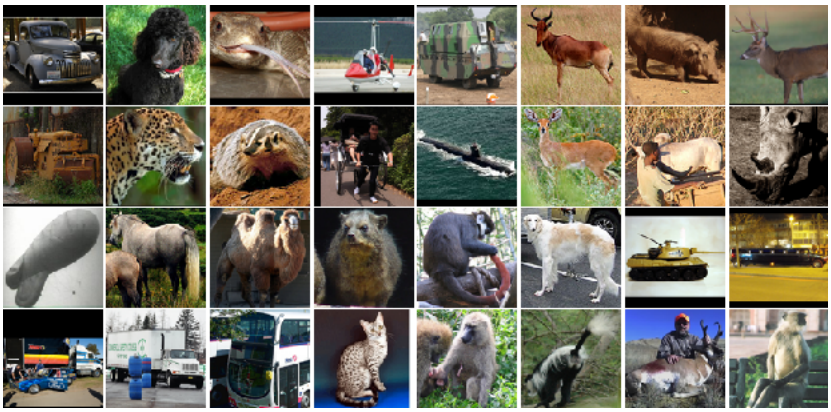


Figure 3-4: Example images from the STL-10 dataset. Every image is of size 96×96 pixels and loosely centered around an object of interest from 10 different classes.

Pre-processing of the Image Data

The STL-10 dataset contains over 100 000 images of size 96×96 pixels with 3 color channels. The dataset was only used during experiments for individual cell classification. In order to integrate its images into the training process, I converted the original images to grayscale and

extracted patches of the center 64×64 pixels from a random 102 400 images from the STL-10 dataset.

The STL-10 dataset was not used for supervised training and I therefore discarded its label data.

3-4 Experiment 1: Classification of Entire Tumor Sections

For the classification of entire tumor sections, I benchmarked the performance of an encoder network that I pre-trained on the TNBC dataset (from now on referred to as the `TNBC` encoder) against two publicly available encoder networks:

- `CIGA` encoder: pre-trained on bright-field microscopy images by Ciga et al. [16] according to the SimCLR algorithm.
- `SIMCLR` encoder: pre-trained on the ImageNet [75] dataset as part of the original SimCLR paper by Chen et al. [14]

Encoder Pre-training

I pre-trained the `TNBC` encoder according to the SimCLR algorithm [14] with parameters similar to those used by Ciga et al. [16] for the `CIGA` encoder:

- ResNet-18 encoder architecture,
- normalized temperature cross-entropy (NT-Xent) loss function with temperature $\tau = 1$,
- batch size $B = 512$ and Layer-wise Adaptive Rate Scaling (LARS) optimizer with learning rate $\eta = 0.3B/256 = 0.6$,
- training for 100 epochs.

For the data source, I used every individual image channel of every image of the TNBC dataset (see Section 3-2-1), meaning that the encoder saw $1010 \times 8 = 8080$ different grayscale images during the pre-training run. Furthermore, I used the following parameters for processing and augmentation of the images (again, based on the parameters used for `CIGA` [16] and `SIMCLR` [14]):

- Crop of a random size corresponding to 1 %–100 % of the original image size (maintaining the original aspect ratio) at a random location,
- resize (and scale) the crop to 224×224 pixels,
- 50 % random flip and rotation augmentation,
- random brightness/contrast variation of up to 80 %.

Classifier Transfer Learning and Linear Evaluation

To achieve the dual objectives of performing linear evaluation of the pre-trained encoders and training classifiers for tumor section classification, I connected the encoders' outputs with simple linear classifiers. Supervised training was performed using the train set as given in Table 3-3 with the following parameters (same as the original SimCLR implementation [14]):

- cross-entropy loss function,
- batch size $B = 64$ and Adam optimizer with learning rate $\eta = 0.1$, weight decay $\lambda = 0.9$ and 0 momentum

During the supervised training of the classifier, the encoder weights remained frozen. Performance was evaluated using classification accuracy as the performance metric on 10 randomly divided batches of the test set, each of equal size.

To account for all 8 channels of an MxIF image, I employed two different configurations for training and testing the classifiers: **Single** (Figure 3-5) and **Siamese** (Figure 3-6).

In the Single configuration (Figure 3-5), each of the 8 image channels is fed individually into the pre-trained encoder. The resulting output feature representation is then used as input for a linear classifier. For every encoder (`TNBC`, `CIGA`, `SIMCLR`), this leads to 8 distinct classifiers, each of which being responsible for classifying the tumor section based on a specific image channel.

In the Siamese configuration (Figure 3-6), the pre-trained encoder is replicated 8 times in parallel, and its outputs are concatenated into a larger feature representation vector. This combined feature representation vector serves as input for a single linear classifier, enabling classification of a tumor section based on all 8 image channels at once.

3-5 Experiment 2: Classification of Individual Cells

For the classification of individual cells, I benchmarked the performance of encoder networks that I pre-trained on the TNBC dataset against each other as I changed their hyperparameters. Essentially, this results in benchmarking against TME-Analyzer, where an increase in performance of an encoder indicates that it is approaching the ability to generate the same labels that Balcioglu et al. obtained with TME-Analyzer [7].

Hyperparameter Optimization in Multiple Rounds

In order to find a network that performs well in classifying individual cells, several hyperparameters need to be determined and optimized. Since the amount of hyperparameters of interest is vast, performing a full grid search optimization would be computationally infeasible. Rather, I split the optimization of network hyperparameters into 4 rounds, where in each round a subset of hyperparameters was optimized and would be used as default parameters when moving to the next round.

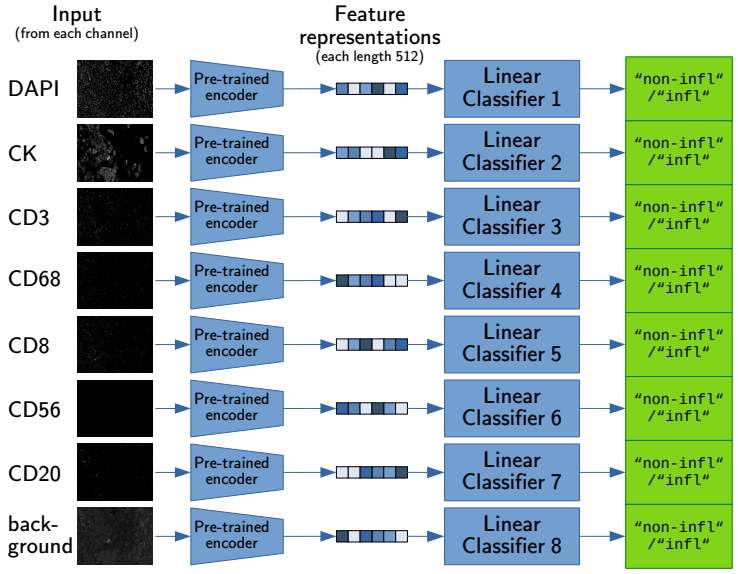


Figure 3-5: Single configuration for tumor section classifier training. In this configuration, 8 separate classifiers are trained per pre-trained encoder (`TNBC`, `CIGA`, `SIMCLR`). Every classifier takes the encoded feature representation of only one image channel as input and learns to assign the class labels `"non-infl"` or `"infl"`. When performing inference on unseen images from the test set, this configuration results in 8 class predictions — one for each image channel.

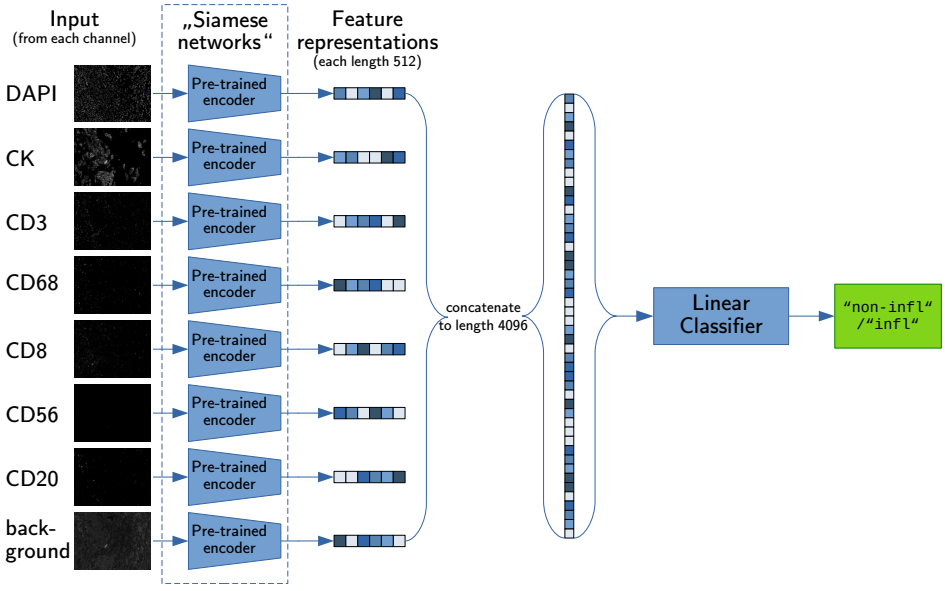


Figure 3-6: Siamese configuration for tumor section classifier training. In this configuration, a single classifier is trained per pre-trained encoder (`TNBC`, `CIGA`, `SIMCLR`). The classifier takes a concatenation of the encoded feature representations of all image channel as input and learns to assign the class labels `"non-infl"` or `"infl"`. When performing inference on unseen images from the test set, this configuration results in one class prediction based on all 8 image channels.

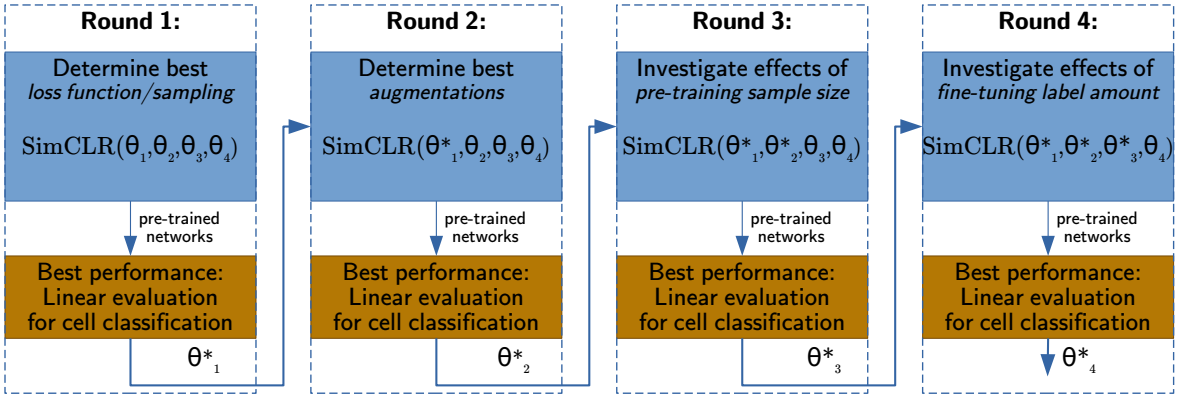


Figure 3-7: Overview of the rounds-based hyperparameter optimization scheme for individual cell classification. In every round, a set of hyperparameters corresponding to a specific aspect of the pre-training or classification is investigated. The hyperparameters that result in the best performance in linear evaluation in one round are carried over to the following round.

An overview of this round-based optimization scheme is shown in Figure 3-7. Here, we consider $\text{SimCLR}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4)$ an abstract function which takes four sets of hyperparameters $\boldsymbol{\theta}_i$ as input and produces a classification network whose performance we want to optimize. The four sets of hyperparameters $\boldsymbol{\theta}_i$ are:

- $\boldsymbol{\theta}_1$: The loss function to use during pre-training and its hyperparameters (such as temperature τ for the NT-Xent loss function), also including a data sampling strategy to improve the convergence of the pre-training loss.
- $\boldsymbol{\theta}_2$: The set of augmentations to use during pre-training.
- $\boldsymbol{\theta}_3$: The datasets and the amount of data to use during pre-training.
- $\boldsymbol{\theta}_4$: The amount of labeled examples to use during supervised training of the linear classifier.

Note that rounds 3 and 4 were not necessarily aimed at finding optimal hyperparameters $\boldsymbol{\theta}_3^\star$ and $\boldsymbol{\theta}_4^\star$, but were rather intended to serve as an investigation of concerns that may arise when implementing the SimCLR algorithm in a practical setting. Namely, "how much and what kind of unlabeled data do we need to provide during unsupervised pre-training, and how much labeled data do we need during supervised training, in order to meaningfully affect performance?"

Within each round, the steps of training a complete classification network were the same as the classification of entire tissue images described in Section 3-4: First, I trained an encoder network via unsupervised pre-training and then connected that encoder to a simple linear classifier, which I trained to assign one or multiple class labels in the transfer learning step.

Encoder Pre-training

I pre-trained encoders according to the SimCLR algorithm [14] with following parameters:

- Residual Network (ResNet)-18 encoder architecture,
- NT-Xent loss function with temperature $\tau = 1$ (**subject to change in round 1**),
- batch size $B = 2048$ and LARS optimizer with learning rate $\eta = 0.3B/256 = 2.4$,
- training for 5 epochs (**subject to change in round 3**),
- 409 600 image patches of size 64×64 (as the completely available amount of 18 360 664 would be too computationally expensive, **subject to change in round 3**),
- 50 % random flip and rotation augmentations (**more were added in round 2**).

Classifier Transfer Learning and Linear Evaluation

Similar to the case of tumor section classification, I connected the encoders' outputs with simple linear classifiers in order to perform individual cell classification and enable linear evaluation of the base encoder. Supervised training was performed using the train set as given in Table 3-4 with the following parameters:

- Binary cross-entropy loss function,
- batch size $B = 2048$ and Adam optimizer with learning rate $\eta = 10^{-4}$, weight decay $\lambda = 10^{-4}$,
- a 30 % dropout layer (this and the previous parameters were experimentally determined to provide a good trade-off between training time and convergence of validation loss),
- 23 684 image patches as described in Table 3-4, which were further split into 80 % training data and 20 % validation data for computation of the validation loss (**subject to change in round 4**),

During the supervised training of the classifier, the encoder weights remained frozen. Performance was evaluated using the classifier's PR-AUC as the performance metric on 11 randomly divided batches of the test set, each of equal size.

For individual cell classification, I only considered the Siamese configuration. The general configuration is the same as the case of tumor section classification and shown in Figure 3-8. Here, the image patches centered around individual cells are fed into the encoder and the linear classifier learns to predict immune phenotype labels for each cell.

3-5-1 Round 1: Loss Function/Sampling

In round 1, I varied several aspects of the loss function used during pre-training as well as the data sampling strategy.

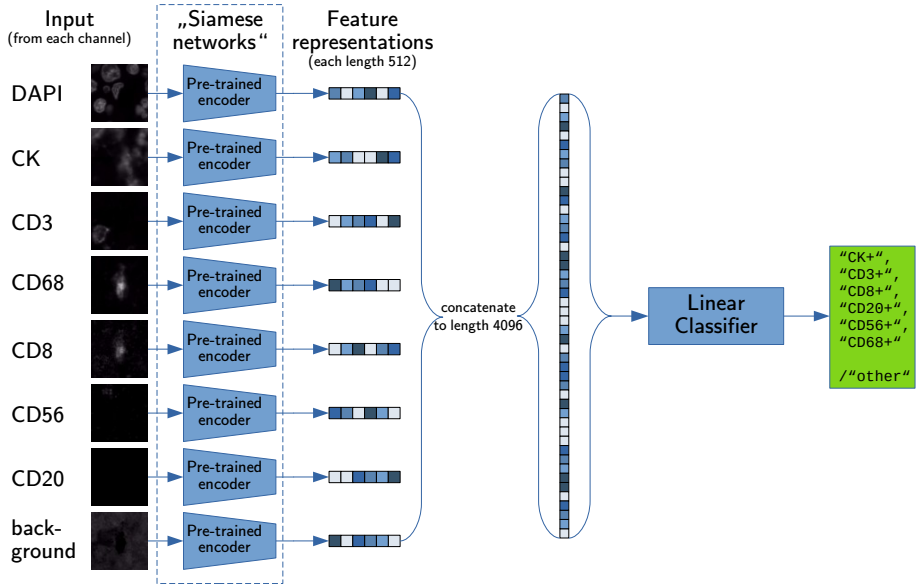


Figure 3-8: Siamese configuration for individual cell classifier training. The general configuration is identical to the Siamese configuration for tumor section classification shown in Figure 3-6. Instead of images of the entire tumor section, smaller patches centered around every cell are fed to the network. The classifier learns to predict one or multiple cell phenotype class labels (note that the `"other"` class label is only assigned if none of the other class labels are assigned).

Varying τ in NT-Xent

Consider the NT-Xent loss function (see Eq. (2-32) in Section 2-4-2, repeated here for convenience):

$$\mathcal{L}^{\mathrm{NT-Xent}} = \underbrace{-\frac{1}{N}\sum_{i,j\in\mathcal{MB}} \mathrm{sim}\left(\mathbf{z}_i, \mathbf{z}_j\right)/\tau}_{=\mathcal{L}_{\mathrm{A}}} \underbrace{+\frac{1}{N}\sum_{i}^{N}\log\left(\sum_{k=1}^{2N}\mathbf{1}_{[k\neq i]}\exp\left(\mathrm{sim}\left(\mathbf{z}_i, \mathbf{z}_k\right)/\tau\right)\right)}_{=\mathcal{L}_{\mathrm{D}}} \tag{3-1}$$

where $\mathcal{L}_{\mathrm{A}}$ is the alignment loss, and $\mathcal{L}_{\mathrm{D}}$ is the distribution loss. I pre-trained encoders using temperature values $\tau \in \{0.05, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0\}$.

Non-contrastive cosine similarity

A simplified loss function can be obtained by simply dropping the distribution loss and using only the alignment loss:

$$\mathcal{L}^{\mathrm{NC}} = \mathcal{L}_{\mathrm{A}} = -\frac{1}{N}\sum_{i,j\in\mathcal{MB}} \mathrm{sim}\left(\mathbf{z}_i, \mathbf{z}_j\right). \tag{3-2}$$

I pre-trained one encoder using this loss function.

NT-Xent with Black Image Marking

The indicator function $\mathbf{1}_{[k \neq i]}$ in the regular NT-Xent loss function prevents the inclusion of $\text{sim}(\mathbf{z}_i, \mathbf{z}_i) = 1$ in the sum of the distribution loss $\mathcal{L}_{\text{D}}$. This indicator function can be adjusted such that it excludes samples based on additional criteria:

$$\mathcal{L}_{\text{D}}^{\text{BM}} = \frac{1}{N} \sum_{i}^{N} \log \left(\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i, \text{img}(i) \notin \text{Black}(t)]} \exp \left(\text{sim} \left(\mathbf{z}_i, \mathbf{z}_k \right) / \tau \right) \right) \tag{3-3}$$

where the new indicator function $\mathbf{1}_{[k \neq i, \text{img}(i) \notin \text{Black}(t)]} = 1$ if and only if $k \neq i$ and $\text{img}(i) \notin \text{Black}(t)$ (i.e. the image corresponding to index i is not considered a black image based on a threshold t). The purpose of this filter is to prevent multiple black images from being considered in the distribution loss, since their computed feature representations would be very similar. To this end, I defined the filter such that $\text{img}(i) \in \text{Black}(t)$ if the brightest pixel in $\text{img}(i)$ is below a threshold $t \in [0, 255]$. I pre-trained encoders using the adjusted loss function $\mathcal{L}^{\text{BM}} = \mathcal{L}_A + \mathcal{L}_{\text{D}}^{\text{BM}}$ with $\tau = 1$ and thresholds $t \in \{10, 20, 30, 40, 50\}$.

NT-Xent with rejection sampling

Using the same filter as described above, I modified the sampling of a pre-training minibatch such that it would only contain one image that fulfills $\text{img}(i) \notin \text{Black}(t)$. In other words, only one black image would be put in the minibatch and all subsequent ones would be rejected and replaced by different images in the dataset. I pre-trained encoders using this sampling strategy with the regular NT-Xent loss function $\mathcal{L}^{\text{NT}-\text{Xent}}$ with $\tau = 1$ and using the same thresholds to determine whether an image is black or not as in the black image marking example: $t \in \{10, 20, 30, 40, 50\}$.

3-5-2 Round 2: Augmentations

In round 2, I varied the augmentations that are applied to the images during the pre-training process. A visual overview of the augmentations is provided in Figure 3-9. They are as follows:

- **Gaussian blur:** Apply Gaussian blurring to the image patch in varying degrees of intensity.
- **Translation:** Shift the center of the image patch in x and y direction, bounded by a radius r of varying length.
- **Zoom:** Zoom in or out of the image patch by zoom factors of varying size.
- **Brightness:** Adjust the brightness of the image patch by varying intensities.
- **Contrast:** Adjust the contrast of the image patch by varying intensities.

In the following, the implementation of these augmentations is elaborated further.

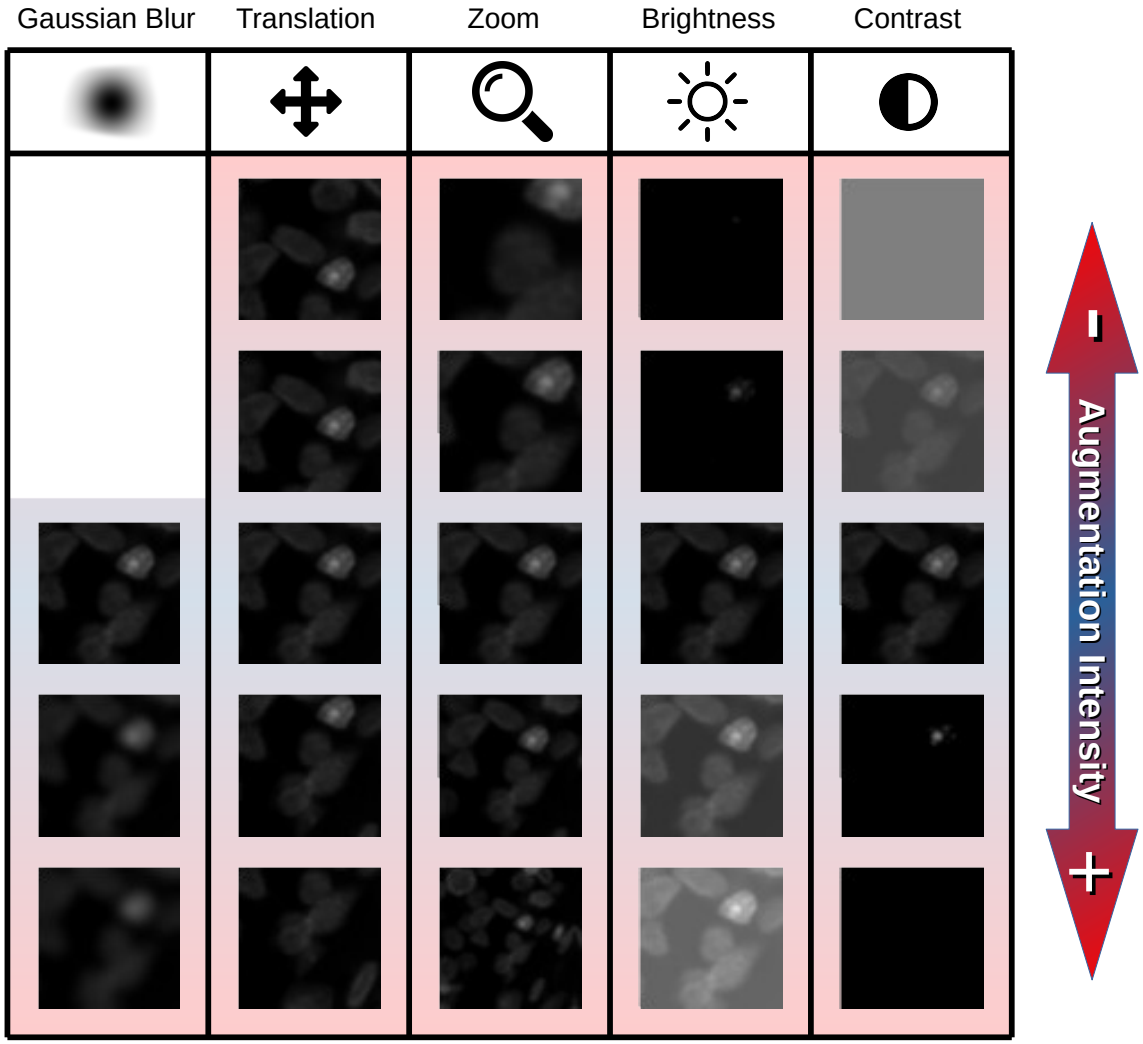


Figure 3-9: Visual overview of the augmentations used during encoder pre-training in round 2. For every column, the third row shows the unaugmented image, and the images above and below show examples of the augmentation applied with positive and negative intensity, respectively.

Gaussian blur

The function for applying Gaussian blur to an image patch depends on a single parameter θ_{gb}, which defines the Gaussian kernel size

$$\mathrm{kernel\_size}(\theta_{\mathrm{gb}}) = 2 \times \theta + 1$$

as well as the standard deviation, which is drawn from a uniform distribution between 0 and θ_{gb}:

$$\mathrm{sigma}(\theta_{\mathrm{gb}}) \sim \mathcal{U}(0, \theta_{\mathrm{b}}).$$

Note that, for sigma $= 0$, no blurring is applied. I pre-trained networks using this augmentation with $\theta_{\mathrm{gb}} \in \{2, 4, 6, 8, 10\}$.

Translation

The function for applying translation to an image patch depends on a single parameter θ_{tr}, which specifies the largest radius ρ that the image patch center may be shifted, as drawn from a uniform distribution:

$$\rho \sim \mathcal{U}(0, \theta_{\mathrm{tr}})$$

Furthermore, a random angle ϕ to specify the translation direction is drawn from a uniform distribution

$$\phi \sim \mathcal{U}(0, 360^{\circ}),$$

in other words, the function translates the image patch center by a distance up to the specified length θ_{tr} in a random direction. Note that, for $\rho = 0$, no translation is applied. I pre-trained networks using this augmentation with $\theta_{\mathrm{tr}} \in \{5, 10, 15, 20, 25\}$. Furthermore, I investigated a special case of this augmentation where, instead of only one, both image patches would be translated by the same radius and in the same direction. I pre-trained networks using this augmentation with $\theta_{\mathrm{tr,ident}} \in \{5, 10, 15, 20, 25, \Delta_{\rho}\}$ where Δ_{ρ} corresponds to a translation to a random location within the tumor tissue image.

Zoom

The function for applying zoom to an image patch depends on a tuple of parameters $(\theta_{\mathrm{zmin}}, \theta_{\mathrm{zmax}})$, which specify the lower and upper bounds, respectively, of the zoom factor Z that is applied to the image patch, as drawn from a uniform distribution

$$Z \sim \mathcal{U}(\theta_{\mathrm{zmin}}, \theta_{\mathrm{zmax}}).$$

Note that, for $Z = 1$, no zoom is applied. I pre-trained networks using this augmentation with $\theta_{\mathrm{zmin}} \in \{0.75, 0.5, 0.25\}$ and $\theta_{\mathrm{zmax}} \in \{1.5, 2, 3, 4\}$.

Brightness

The function for applying brightness adjustment to an image patch depends on a single parameter θ_b, which specifies the lower and upper bound of the brightness modifier B, as drawn from a uniform distribution

$$B \sim \mathcal{U}(-\theta_b, \theta_b).$$

Note that, for $B = 0$, no brightness adjustment is applied, for $B = -1$, the image patch is made as dark as possible and that, for $B = 1$, the image is made as bright as possible. I pre-trained networks using this augmentation with $\theta_b \in \{0.25, 0.5, 0.75, 1\}$.

Contrast

The function for applying contrast adjustment to an image patch depends on a single parameter θ_c, which specifies the lower and upper bound of the contrast modifier C, as drawn from a uniform distribution

$$C \sim \mathcal{U}(-\theta_c, \theta_c).$$

Note that, for $C = 0$, no contrast adjustment is applied, for $C = -1$, the image patch contrast is made as low as possible and that, for $C = 1$, the image patch contrast is made as high as possible. I pre-trained networks using this augmentation with $\theta_c \in \{0.25, 0.5, 0.75, 1\}$.

3-5-3 Round 3: Pre-training Sample Size

In round 3, I varied the amount of data that was used during the unsupervised pre-training and added images from foreign datasets (CellPose and STL-10)

Up to this point, the amount of image patches used from the TNBC dataset during pre-training was 409 600 (corresponding to 51 200 individual cells). I tested the effect of lowering and increasing this amount by multiplying the amount of images by $1/n$ and n, respectively, and pre-training encoders with $n \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

For testing the effect of images from foreign data sources, I included the CellPose (Section 3-3-1) and STL-10 (Section 3-3-2) datasets during pre-training in the following combinations

- TNBC only $\rightarrow N_{\text{total}} = 102\,400$
- CellPose only $\rightarrow N_{\text{total}} = 102\,400$
- STL-10 only $\rightarrow N_{\text{total}} = 102\,400$
- TNBC ($N = 102\,400$) &
  CellPose ($N = 102\,400$) $\rightarrow N_{\text{total}} = 204\,800$
- TNBC ($N = 102\,400$) &
  STL-10 ($N = 102\,400$) $\rightarrow N_{\text{total}} = 204\,800$
- CellPose ($N = 102\,400$) &
  STL-10 ($N = 102\,400$) $\rightarrow N_{\text{total}} = 204\,800$

- TNBC ($N = 204\,800$) &
  CellPose ($N = 102\,400$) &
  STL-10 ($N = 102\,400$) $\rightarrow N_{\text{total}} = 409\,600$

The images used per dataset were identical for every combination. For the last combination, I sampled an additional 102 400 images for TNBC.

3-5-4 Round 4: Supervised Training Label Amount

In round 4, I varied the amount of labeled data that was available during the supervised transfer learning. The balanced training set used for the supervised transfer learning, shown in Table 3-4 contains all cells with the `"CD56+"` label present in the total training set (see Table 3-4). Effectively, this means that the supervised classifiers trained on this balanced training set are using 100 % of the available label data. In order to study the effects of classifier performance when only a fraction of labeled data is available, I re-balanced this training set such that no class label occurs more than an amount corresponding to P, where $P \in \{100, 90, 80, 70, 60, 50, 40, 30, 20, 10\}$ % is the percentage of the amount of class labels in relation to the unbalanced training set.

The proportions of cells in relation to the unbalanced training set are shown in Table 3-5. Here, the column for "$\leq$**100%**" corresponds to the balanced train set specified in Table 3-4. For every re-balancing, whenever a certain cell was removed from the dataset, a cell of the `"other"` class was added back in, such that the total amount of cells in every set remained at 23684.

In a separate experiment, I deviated from the previously defined train/test split and constructed three new splits according to the patient-level spatial phenotype labels (see Table 3-1): In each set, only cells from patients with a specific spatial phenotype were put in the training set and all remaining cells were put in the test set. I then trained three classifiers using each of these training sets.

In this round, I did not pre-train any new encoders. Instead, all classifiers were trained on top of the best-performing encoder from round 3.

Table 3-5: Summary of label percentages per re-balanced training set for the task of individual cell classification. The percentages in every column specify the upper limit of the amount that any class label may occur in the re-balanced set. For example, for the re-balanced training set specified by the column under "$\leq$**60%**", no class label may occur more than 60 % in relation to the entire training set. Note that this re-balancing primarily affects cells with the `"CD56+"` label, which is by far the rarest label in the TNBC dataset. Whenever cells were removed from the dataset due to this re-balancing, an equal amount of cells with the `"other"` label were added back in such that the total amount of cells in every re-balanced training set remained at 23684.

Label	$\leq$**100%**	$\leq$**90%**	$\leq$**80%**	$\leq$**70%**	$\leq$**60%**	$\leq$**50%**	$\leq$**40%**	$\leq$**30%**	$\leq$**20%**	$\leq$**10%**
`"CK+"`	0.89 %	0.88 %	0.88 %	0.87 %	0.87 %	0.87 %	0.87 %	0.86 %	0.86 %	0.86 %
`"CD3+"`	4.30 %	4.28 %	4.27 %	4.25 %	4.22 %	4.21 %	4.20 %	4.18 %	4.17 %	4.16 %
`"CD8+"`	3.03 %	3.02 %	3.01 %	3.00 %	2.99 %	2.99 %	2.98 %	2.97 %	2.96 %	2.84 %
`"CD20+"`	5.51 %	5.50 %	5.50 %	5.48 %	5.46 %	5.46 %	5.44 %	5.44 %	5.43 %	5.44 %
`"CD56+"`	100.00 %	89.87 %	79.84 %	69.86 %	59.79 %	50.00 %	39.89 %	29.93 %	19.92 %	9.98 %
`"CD68+"`	12.24 %	12.25 %	12.25 %	12.25 %	12.27 %	12.28 %	12.31 %	12.32 %	12.32 %	10.02 %
`"other"`	0.51 %	0.54 %	0.57 %	0.60 %	0.62 %	0.65 %	0.68 %	0.70 %	0.73 %	0.81 %

Chapter 4

Results

This chapter presents the results of the conducted experiments outlined in Chapter 3. It is structured in a way such that each section presents results that are relevant to the research questions:

- *How can a self-supervised learning model be designed to learn features from multiplex immunofluorescence (MxIF) images with an arbitrary number of color channels?*
  - This question is addressed in Section 4-1, which shows the effectiveness of the Siamese architecture to facilitate learning of features in MxIF images.
- *What are the hyperparameters for such a model that improve feature learning of individual cells in MxIF images?*
  - This question is addressed in Section 4-2, which lists the hyperparameters obtained after the rounds-based optimization scheme, as well as notable results related to these hyperparameters.
- *Can the features learned by such a model improve label-efficiency for the task of individual cell classification?*
  - This question is addressed in Section 4-3, which shows results related to how the amount and type of labels used during supervised training affects performance in the cell classification task.

4-1 Self-Supervised Learning Model for Multi-Channel MxIF Image Feature Extraction

In this section, the effectiveness of a self-supervised learning model using the SimCLR algorithm and Siamese configuration for feature extraction from multi-channel MxIF images is explored. The following subsection presents the results of the experiments for the entire tumor section classification task. All results shown are computed as an ensemble across 10 batches of the test set (see Table 3-3).

4-1-1 Effective Feature Learning from MxIF Images with Multiple Color Channels using SimCLR and Siamese Configuration for Tumor Classification

Figure 4-1 shows the accuracy scores achieved by encoder/classifier pairs trained in the Single configuration (see Figure 3-5). For all encoders, the classifiers trained on feature representations from the CD3 or CD8 image channels were able to achieve performance higher than most other channels. This is in accordance with the fact that these channels give the best information regarding whether a tumor is inflamed or not [4]. The `TNBC` encoder resulted in overall best performance, with the most useful features being learned from the DAPI channel for a median accuracy of 0.913.

Notably, the `SIMCLR` encoder achieved the second best performance, with the most useful features being generated from the CD8 channel for a median accuracy of 0.898. Despite the encoder not being trained on microscopy images, it provides useful features for this classification task.

The `CIGA` encoder achieved the worst performance. In the figure, `CIGA_INV` represents the same encoder, but it was provided with color-inverted images to make them visually more similar to the bright-field images that the `CIGA` encoder was pre-trained on. In both cases, the CD8 channel generated the most useful features, resulting in median accuracy scores of 0.845 for both variants. In the color-inverted case, the CD3 channel also generated slightly better representations resulting in an accuracy score of 0.83, compared to 0.786 in the non-inverted case.

Figure 4-2 shows the accuracy scores achieved by encoder/classifier pairs trained in the Siamese configuration (see Figure 3-6). Using this configuration, performance according to the median accuracy scores improved for `TNBC` with 0.937 (+2.63 %), for `CIGA` with 0.864 (+2.25 %), for `CIGA_INV` with 0.879 (+4.02 %), but deteriorated for `SIMCLR` with 0.864 (−3.79 %) when compared to the best-performing classifiers in the Single configuration.

The fact that the `TNBC` encoder obtains the best performance indicates that the SimCLR algorithm can be used to learn useful feature representations from MxIF images. Furthermore, the Siamese configuration provides a recipe for a network architecture that can learn to classify images with an arbitrary amount of color channels.

4-2 Hyperparameters for Improved Feature Learning of Individual Cells in MxIF Images

In this section, the results related to the hyperparameter optimization are presented. The optimal hyperparameters obtained in each round of the optimization scheme described in Section 3-5 are as follows:

- **Round 1 (Loss function/data sampling)**: normalized temperature cross-entropy (NT-Xent) loss function with temperature parameter $\tau = 0.05$ combined with black image marking using a maximum pixel intensity threshold of $t = 30$.
- **Round 2 (Augmentations)**: Maximum zoom-out of $\times 2$ and brightness and contrast adjustment of 50 %. No translation, zoom-in, and Gaussian blur.

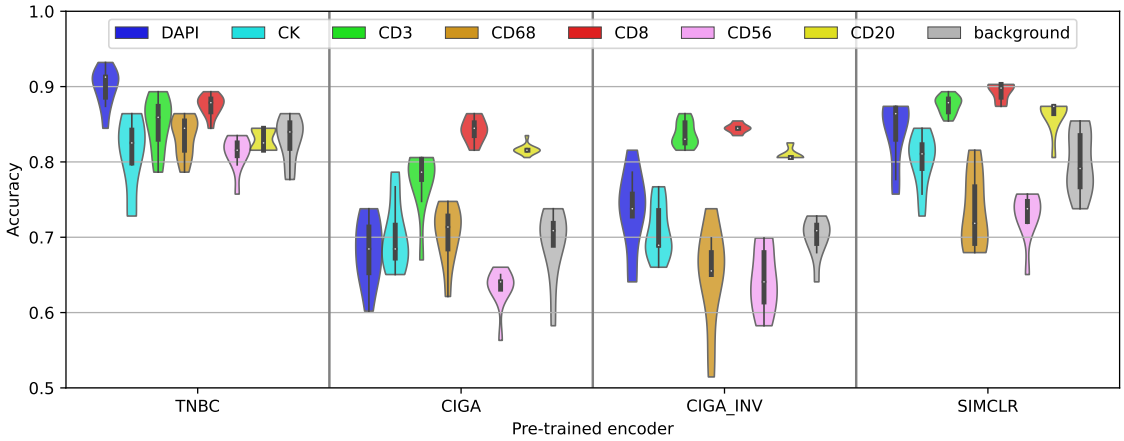


Figure 4-1: Violin plot of the classification accuracy scores for classifiers trained in the single configuration. The `TNBC` encoder has been pre-trained on the TNBC dataset, the `CIGA` encoder [16] has been pre-trained on bright-field microscopy images and the `SIMCLR` encoder [14] has been pre-trained on ImageNet [76]. Note that `CIGA_INV` is the same encoder as `CIGA`, but it was fed with color-inverted images to make them more similar to bright-field images. Each colored violin represents an encoder that has been trained on top of the respective encoder and its color denotes which image channel it was trained on. All encoders were able to provide useful features using at least either the CD3 or CD8 color channels for tumor classification between inflamed and non-inflamed. Notably, the `TNBC` encoder provided particularly good features using the DAPI channel, indicating that it learned meaningful spatial patterns in the tumor. Results are computed as an ensemble across 10 batches of the test set. Note that, due to the train/test split composition, an accuracy score of 0.65 is as good as a random guess.

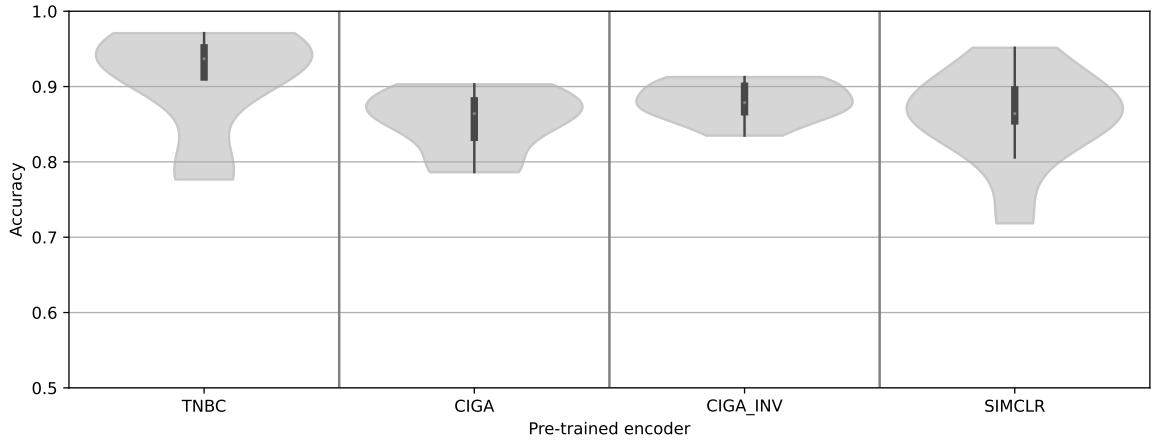


Figure 4-2: Violin plot of the classification accuracy scores for classifiers trained in the Siamese configuration. The `TNBC` encoder has been pre-trained on the TNBC dataset, the `CIGA` encoder [16] has been pre-trained on bright-field microscopy images and the `SIMCLR` encoder [14] has been pre-trained on ImageNet [76]. Note that `CIGA_INV` is the same encoder as `CIGA`, but it was fed with color-inverted images to make them more similar to bright-field images. The Siamese configuration resulted in an improvement of performance for all encoder/classifier pairs. The `TNBC` encoder learned particularly useful features which resulted in it clearly outperforming the other encoders that were not pre-trained on the TNBC dataset. Results are computed as an ensemble across 10 batches of the test set. Note that, due to the train/test split composition, an accuracy score of 0.65 is as good as a random guess.

- **Round 3 (Pre-training datasets)**: In general, more pre-training data of the triple-negative breast cancer (TNBC) dataset lead to better performance, with the maximum amount of images tested being 4 096 000. No performance improvement from including CellPose and STL-10 datasets during pre-training.
- **Round 4 (Supervised training label availability)**: The initially constructed balanced dataset, corresponding to an inclusion of 100 % of `"CD56+"` class labels, resulted in best performance, but larger datasets for supervised training were not tested. For the particular case of training on only cells of one kind of tumor phenotype, with the remaining ones serving as test set, using cells from the excluded tumor phenotype resulted in best performance (see Section 4-3-2).

In the following subsections, notable results observed during the experiments for the individual cell classification task are presented. All results shown are computed as an ensemble across 11 batches of the test set (see Table 3-4).

4-2-1 SimCLR Loss Function has the Largest Impact on Feature Quality

Figure 4-3 shows the Area Under the Precision-Recall Curve (PR-AUC) scores after every round of the hyperparameter optimization scheme explained in Section 3-5. Here, `round0` refers to the beginning of the optimization scheme. Overall, the MoM(PR-AUC) increased from 0.34 at the beginning of the optimization scheme to 0.75 at the end of round 3 (the results of round 4 are shown separately in Section 4-3-1). As we can see, the jump in mean

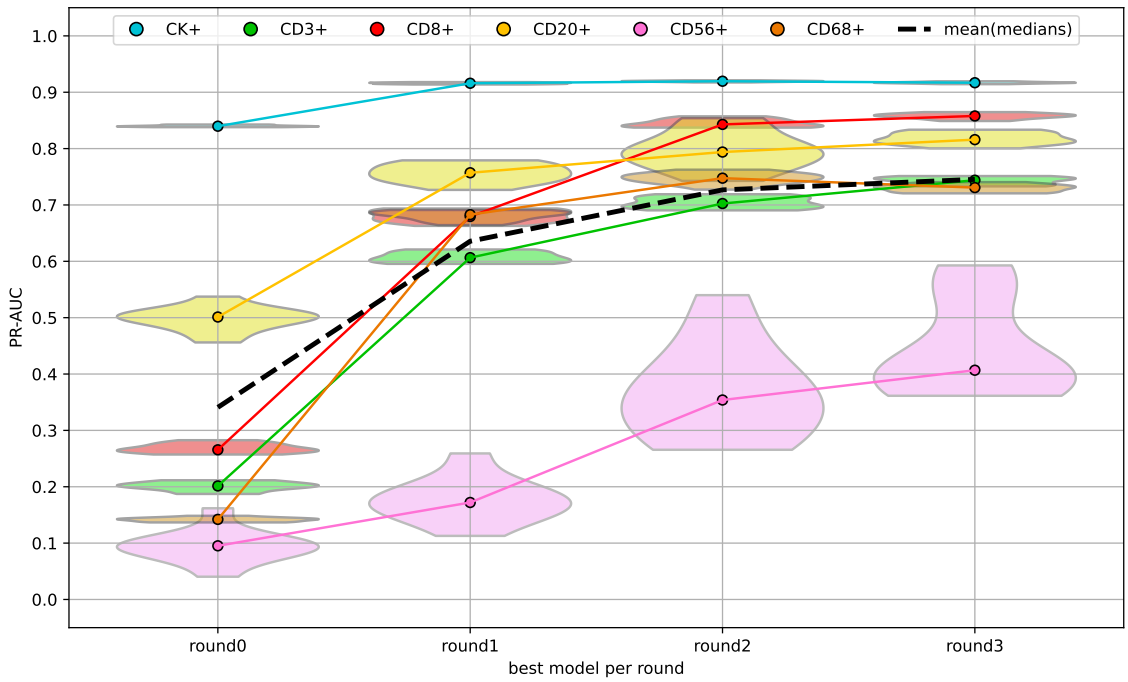


Figure 4-3: Violin plot of the PR-AUC scores for the best encoders obtained in the round-based hyperparameter optimization scheme. Note that the displayed results are of the encoders obtained at the end of the specified round and `round0` refers to the beginning of the optimization — before any hyperparameters have been changed. `round1` affected hyperparameters related to the pre-training loss function, `round2` affected hyperparameters related to pre-training augmentations, and `round3` affected pre-training dataset size. Results are computed as an ensemble across 11 batches of the test set. The colored markers for each cell phenotype denote the median of the distribution of scores across the 11 batches. The dashed black line denotes the mean of all individual class medians (MoM(PR-AUC)).

performance after finishing round 1 is the largest, indicating that tuning of the loss function has the biggest impact on feature quality of the encoders. Note, however, that features for the `"CD56+"` class were more affected by optimizing the pre-training augmentations in round 2 where the PR-AUC(`"CD56+"`) increased by 0.18 compared to an increase of 0.08 after round 1. The reason for this can be seen in Figure 4-4, which shows the PR-AUC scores for varying values of τ in the standard NT-Xent loss function: values of $\tau < 1$ heavily improved feature quality for all classes except `"CD56+"`, which instead improved more strongly for values of $\tau > 1$. Having to make a trade-off, I chose $\tau = 0.05$ since the improvement in mean performance was larger overall.

Figure 4-5 shows the PR-AUC scores for the four different loss functions and sampling strategies explained in Section 3-5-1. Here, all scores are for $\tau = 1$ and the best-performing brightness thresholds for the black image marking ($t = 20$) and rejection sampling ($t = 30$). Overall, the non-contrastive loss function (ntxentNC) performed the worst with a MoM(PR-AUC) of 0.25, whereas the NT-Xent loss function with black marking and intensity threshold of 30 performed best with a MoM(PR-AUC) of 0.39.

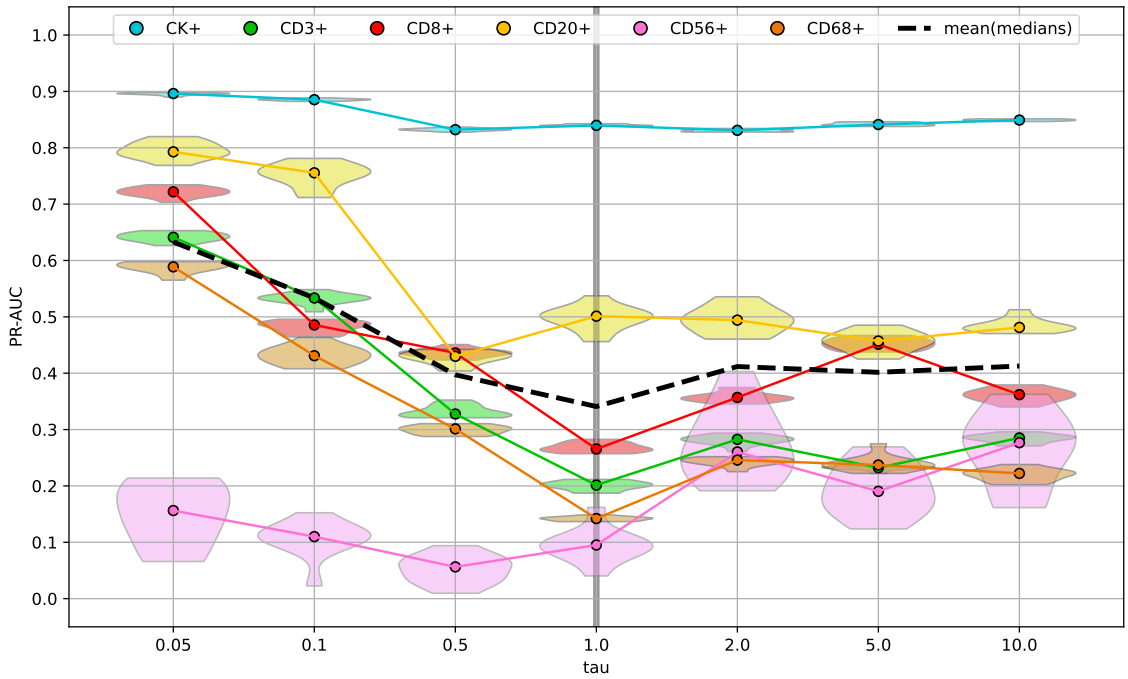


Figure 4-4: Violin plot of the PR-AUC scores for varying values of the temperature parameter τ of the NT-Xent loss function. We can observe a change in τ having a different effect on performance for different cell phenotypes. Notably, PR-AUC("CD56+") increases more strongly for $\tau > 1$ whereas all other PR-AUCs increase more strongly for $\tau < 1$. Results are computed as an ensemble across 11 batches of the test set. The colored markers for each cell phenotype denote the median of the distribution of scores across the 11 batches. The dashed black line denotes the mean of all individual class medians (MoM(PR-AUC)). The vertical gray line denotes the default value at the start of the optimization.

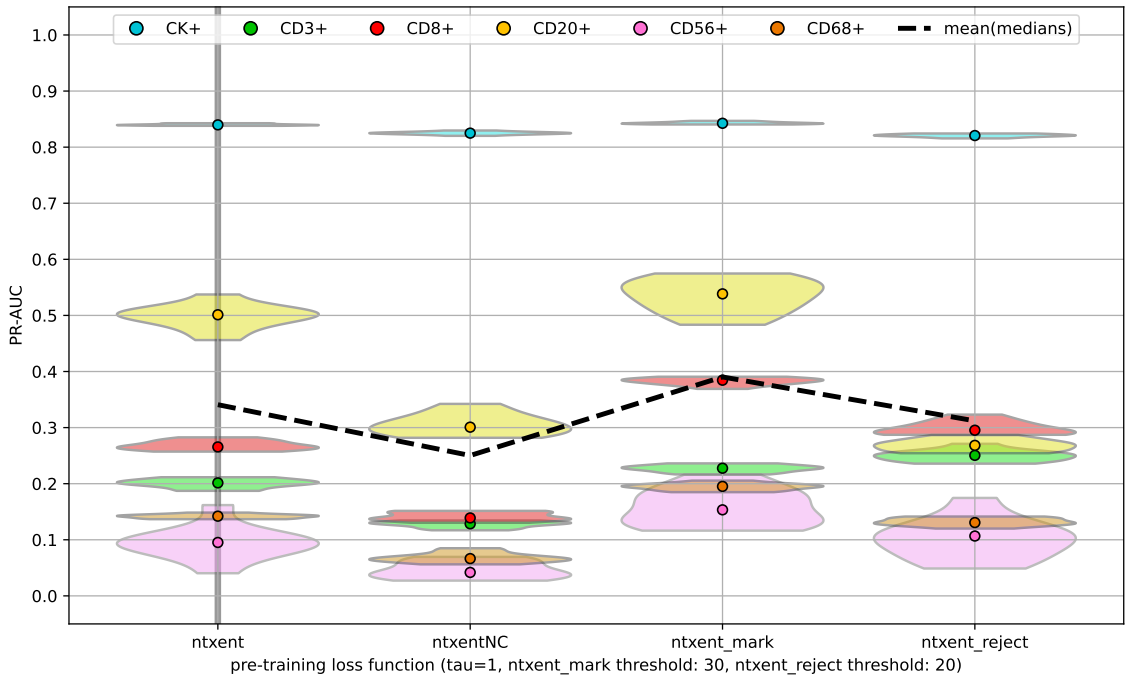


Figure 4-5: Violin plot of the PR-AUC scores for various pre-training loss functions. `ntxent` is the standard NT-Xent loss function, `ntxentNC` is only the alignment loss term of the NT-Xent loss function, `ntxent_mark` is the NT-Xent loss function with a black image marking term based on a pixel intensity threshold and `ntxent_reject` is the standard NT-Xent loss function, but utilizing rejection sampling based on a pixel intensity threshold. Results are computed as an ensemble across 11 batches of the test set. The colored markers for each cell phenotype denote the median of the distribution of scores across the 11 batches. The dashed black line denotes the mean of all individual class medians (MoM(PR-AUC)). The vertical gray line denotes the default value.

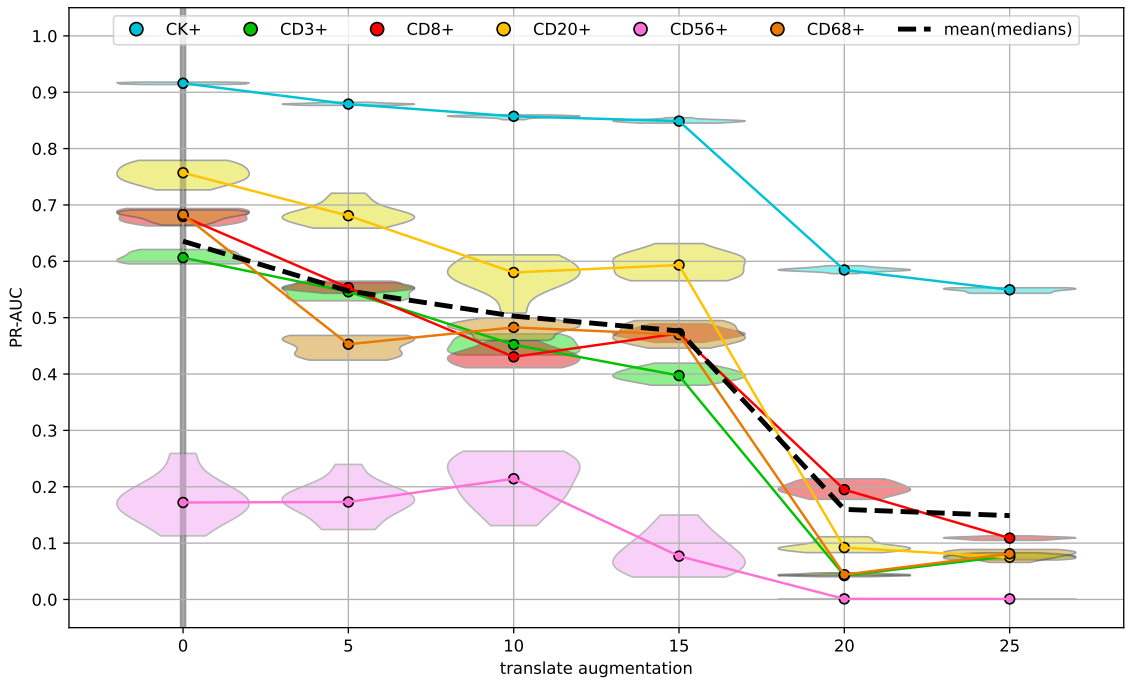


Figure 4-6: Violin plot of the PR-AUC scores for varying intensities of the translation augmentation applied during unsupervised pre-training. In general, any amount of translation negatively impacted the quality of the learned features. An exception are cells with the "CD56+" class label, where performance was unaffected or even increased with translation distances of up to 10 pixels. A harsh drop can be observed for distances larger than 15 pixels, indicating that the encoders require the contents of the augmented image pairs to be spatially coherent in order to learn good features. Results are computed as an ensemble across 11 batches of the test set. The colored markers for each cell phenotype denote the median of the distribution of scores across the 11 batches. The dashed black line denotes the mean of all individual class medians (MoM(PR-AUC)). The vertical gray line denotes the default value at the start of the optimization.

4-2-2 Cell Locations are not Required during Pre-training to Learn High-quality Feature Representations

Figure 4-6 shows the PR-AUC scores obtained from varying the translation augmentation during pre-training (see Section 3-5-2). The translation augmentation has an exclusively negative effect on the MoM(PR-AUC), which ranges from 0.636 to 0.476 between translation radii of 0 and 15, respectively. With radii larger than that, the MoM(PR-AUC) values harshly dropped to around 0.15. This indicates that the encoders have a preference for positional coherence of the center of the augmented image pairs that are compared during pre-training.

Figure 4-7 shows the PR-AUC scores obtained from varying the translation augmentation using the special case described in Section 3-5-2: both image patches in the augmented image pair were shifted by the same distance and in the same direction, including a shift to a random location within the tumor tissue image. As we can see, this augmentation has a minimal impact on the PR-AUC scores and the MoM(PR-AUC) remains relatively constant around values of 0.63–0.64. This indicates that the encoder does not necessarily require image patches centered around cells in order to learn high-quality feature representations for the

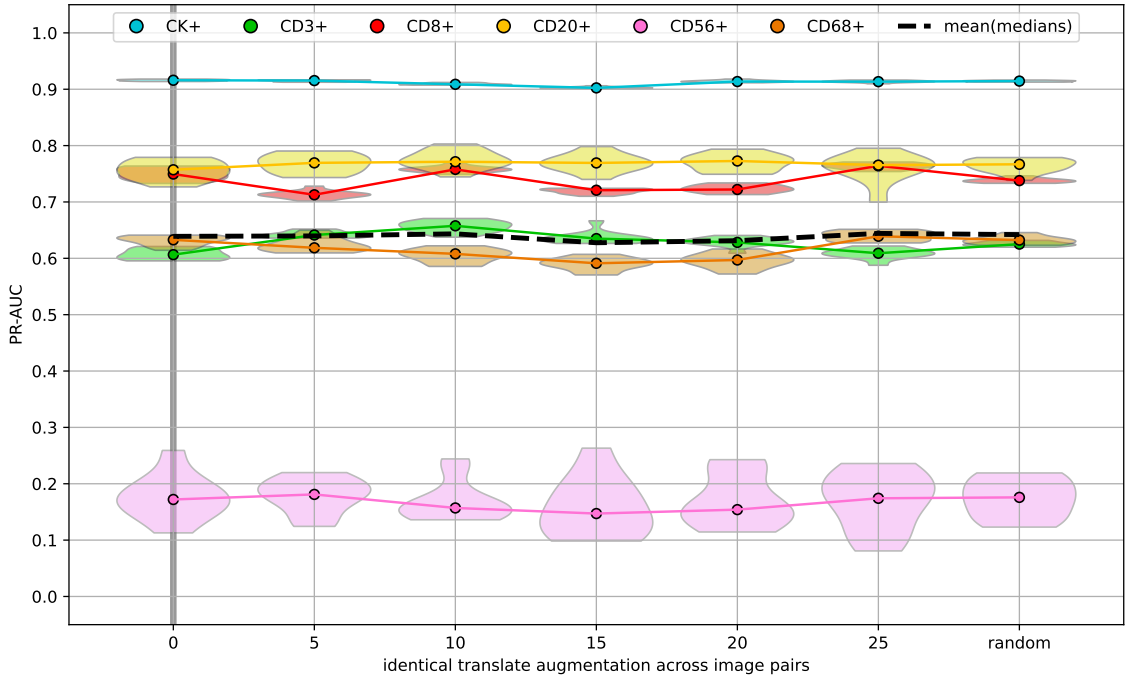


Figure 4-7: Violin plot of the PR-AUC scores for varying intensities of the special translation augmentation applied during unsupervised pre-training. This translation augmentation shifted both images of the augmented image pair by the same distance and in the same direction. The x-tick labeled `random` indicates that the center of the image patch was shifted to a random location in the tumor tissue image. The resulting scores a largely invariant to this type of augmentation, indicating that the encoder does not actually require image patches to be centered around cells in order to learn useful features during pre-training. Results are computed as an ensemble across 11 batches of the test set. The colored markers for each cell phenotype denote the median of the distribution of scores across the 11 batches. The dashed black line denotes the mean of all individual class medians (MoM(PR-AUC)). The vertical gray line denotes the default value at the start of the optimization.

purpose of cell classification.

4-2-3 Longer Pre-training Improves Feature Quality for CD56+ and CD20+ Cells

Figure 4-8 shows the effect of pre-training encoders on a varying amount of unlabeled data. Specifically, the amount of cells ranged from 5120 ($\times 1/10$) to 512 000 ($\times 10$) ($N_{\text{img}} = 40\,960$ to $N_{\text{img}} = 4\,096\,000$ for the corresponding amount of image patches). In this experiment, only the amount of images was varied and the amount of training steps was kept constant. We can see a slight upward trend in MoM(PR-AUC) from 0.712 when using only 1/10 of the dataset to 0.749 (+5.2 %) when using 10 times the base amount. This trend is largely influenced by PR-AUC(`"CD56+"`) scores, which range from 0.319 to 0.42 (+31.7 %), respectively. Another slight improvement can be seen in PR-AUC(`"CD20+"`) scores, which range from 0.759 to 0.811 (+6.85 %), respectively. Although an upward trend is visible for the PR-AUC(`"CD56+"`) and PR-AUC(`"CD20+"`) medians, their variance fluctuates strongly.

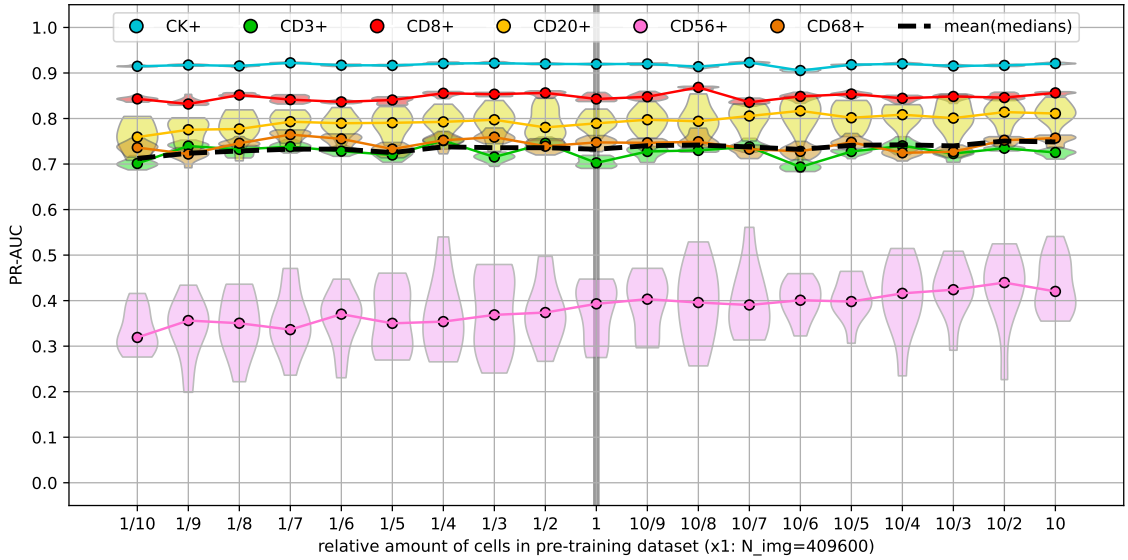


Figure 4-8: Violin plot of the PR-AUC scores for varying the amount of unlabeled data available during unsupervised pre-training. The values on the x-axis denote the relative amount to the default amount of images, which was 409 600 in this case. Upward trends for PR-AUC("CD56+") and PR-AUC("CD20+") are noticeable with an increase in pre-training data, but variance remains high overall for the "CD56+" class. Results are computed as an ensemble across 11 batches of the test set. The colored markers for each cell phenotype denote the median of the distribution of scores across the 11 batches. The dashed black line denotes the mean of all individual class medians (MoM(PR-AUC)). The vertical gray line denotes the default value.

4-2-4 Inclusion of CellPose and STL-10 Datasets during Pre-training does not Improve Feature Quality

Although pre-training on more samples from the TNBC dataset resulted in better learned feature representations (see Section 4-2-3), the inclusion of images from the CellPose dataset (Section 3-3-1) and STL-10 dataset (Section 3-3-2) during pre-training did not improve the quality of learned feature representations according to the MoM(PR-AUC).

Figure 4-9 shows the PR-AUC scores from pre-training encoders on various combinations of these datasets with amounts of images $N_{\text{img}} \in \{102\,400, 204\,800, 409\,600\}$. In every case, performing pre-training on only the TNBC dataset resulted in the best performance. However, some performance scores are notable. The encoder pre-trained on 102 400 images from the STL-10 dataset resulted in a PR-AUC(`"CD56+"`) of 0.37 compared to the 0.39 achieved by the encoder pre-trained on the same amount of images from the TNBC dataset. While the PR-AUC(`"CD56+"`) of 0.23 from the encoder pre-trained on 102 400 images from the CellPose dataset is not nearly as high, its PR-AUC(`"CK+"`) of 0.89 is close to the 0.91 achieved by the encoder pre-trained on TNBC. A similar effect can be observed in the case of 204 800 pre-training images: the encoder pre-trained on an equal amount of TNBC and CellPose images achieved a PR-AUC(`"CK+"`) of 0.9 and the encoder pre-trained on only TNBC images achieved a PR-AUC(`"CK+"`) of 0.91.

4-3 Improving Label-efficiency for Individual Cell Classification

The results in the following subsections are from experiments for the individual cell classification task, using the best encoder network (see hyperparameters in Section 4-2) and only considering changes in the labeled data available during the supervised training. All results shown are computed as an ensemble across 11 batches of the test set (see Table 3-4).

4-3-1 SimCLR Improves Label-efficiency for Individual Cell Classification

Figure 4-10 shows the PR-AUC scores obtained from performing the experiment for round 4 (see Section 3-5-4 and Table 3-5). Since `"CD56+"` is the rarest class where 100 % of labels is used, compared to 12.24 % of labels used for the second rarest `"CD68+"` class, the imposed limit on the label percentage primarily affects the `"CD56+"` class.

During the supervised transfer learning of the classifier, the amount of available labels could be decreased down to 40 % before the PR-AUC(`"CD56+"`) dropped from about 0.4 between 100 % and 40 % to 0.26 (−35 %) at 10 % and the MoM(PR-AUC) from 0.73 to 0.71 (−2.74 %), respectively.

These results can be compared to the performance by the classifiers (with 100 % labels available) based on the best encoder networks obtained after rounds 1 and 2 as shown in Figure 4-3:

- With only 10 % available labels, the best encoder from round 1 is outperformed in MoM(PR-AUC) with 0.71 to 0.64 (−9.86 %) and in PR-AUC(`"CD56+"`) with 0.26 to 0.17 (−34.62 %).

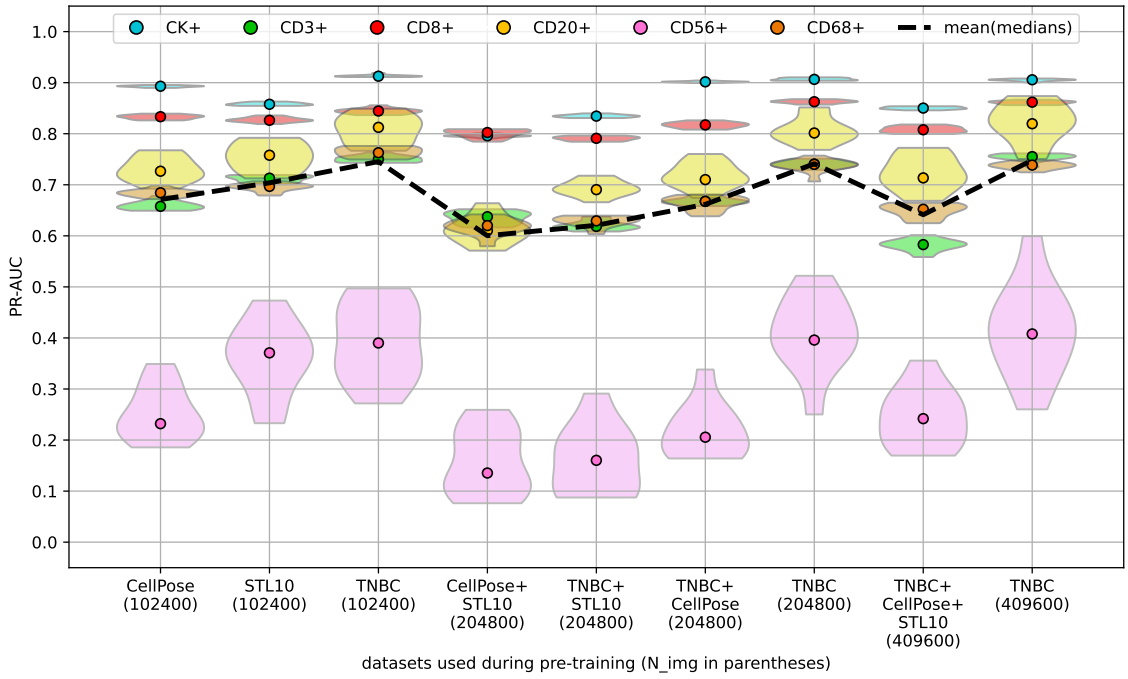


Figure 4-9: Violin plot of the PR-AUC scores for various combinations of data sources for the pre-training datasets. Datasets included were the TNBC, CellPose and STL-10 datasets. The numbers in parentheses denote the amount of images comprising the datasets. In case multiple data sources were included, image amounts were split evenly — except for the case of `TNBC+CellPose+STL10` where the former was represented with 204 800 images and the latter two with 102 400 images each. Usage of only the TNBC dataset resulted in best performance overall, but the encoder pre-trained on only the STL-10 dataset learned relatively good features for the `"CD56+"` class and inclusion of the CellPose dataset alone or in combination with TNBC resulted in an improvement of learned features for the `"CK+"` class. Results are computed as an ensemble across 11 batches of the test set. The colored markers for each cell phenotype denote the median of the distribution of scores across the 11 batches. The dashed black line denotes the mean of all individual class medians (MoM(PR-AUC)).

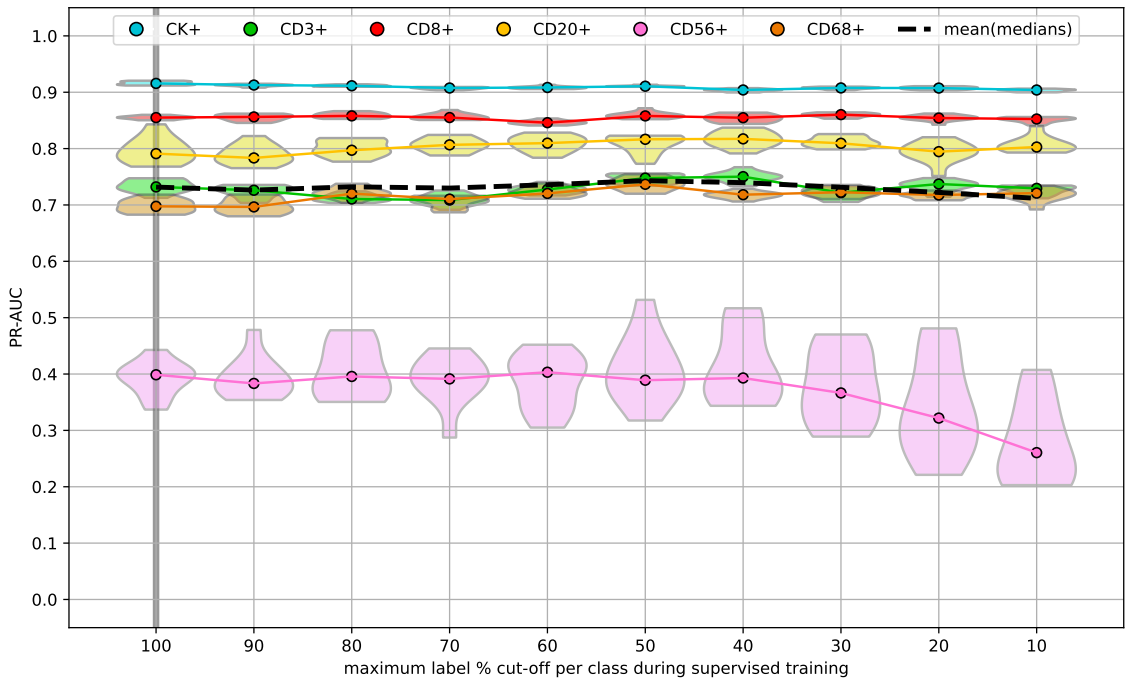


Figure 4-10: Violin plot of the PR-AUC scores for varying the amount of labeled data available during supervised transfer learning. The percentages on the x-axis denote an upper limit to the amount of cells that was included in the respective dataset. In practice, only cells with the `"CD56+"` and `"CD68+"` labels were affected by this limitation. Results are computed as an ensemble across 11 batches of the test set. The colored markers for each cell phenotype denote the median of the distribution of scores across the 11 batches. The dashed black line denotes the mean of all individual class medians (MoM(PR-AUC)). The vertical gray line denotes the default value.

- With only 30 % available labels, the best encoder from round 2 is outperformed in MoM(PR-AUC) with 0.731 to 0.727 (−0.55 %) and in PR-AUC(`"CD56+"`) with 0.366 to 0.354 (−3.28 %).

This indicates that the encoders which received more hyperparameter optimization improve the label-efficiency of a classifier trained on top of them.

4-3-2 Supervised Training on "Excluded" Tumor Phenotype Results in Best Generalization of Classification Performance

Figure 4-11 shows the PR-AUC scores obtained from training classifiers only on cells from patients with the spatial phenotype labels `"infl"`, `"excl"`, and `"ign"` and testing the classifier on the remaining two spatial phenotypes. The resulting PR-AUC scores give an insight into how well a cell classifier trained on a certain type of tumor generalizes to other tumors. Here, the classifier trained on cells from tumors of the excluded phenotype achieves the highest MoM(PR-AUC) of 0.719, the one trained on the inflamed phenotype achieves 0.677 and the one trained on the ignored phenotype achieves 0.632. Notable are the PR-AUC(`"CD56+"`) scores which are 0.448 for excluded, 0.208 for ignored, and 0.144 for inflamed.

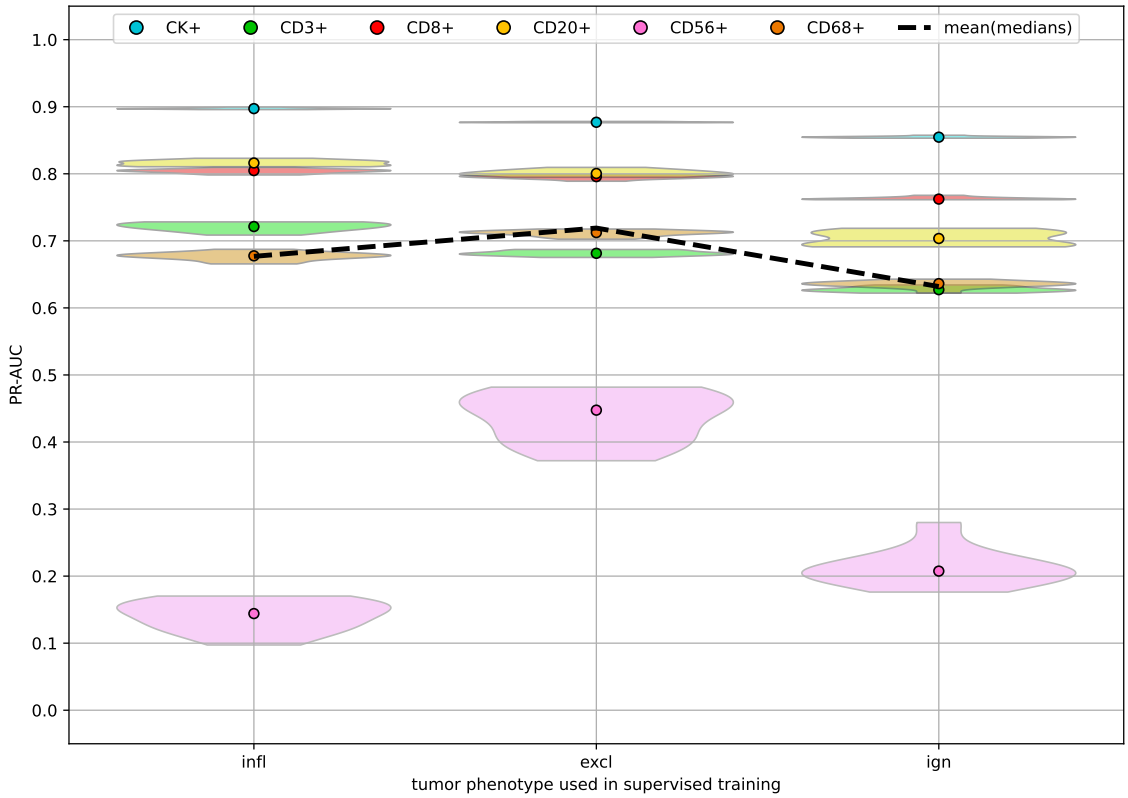


Figure 4-11: Violin plot of the PR-AUC scores for various combinations of supervised train/test splits according to patient-level tumor phenotypes. The x-tick labels denote which tumor phenotype was used for the training set, whereas the other two were used for the test set (e.g. `"ign"` indicates that the classifier was trained on cells belonging to tumors of the ignored phenotype and was tested on cells from the inflamed and excluded phenotypes). The scores indicate that training on cells of excluded phenotype tumors, in general, results in the best generalization performance when testing on cells from other tumor phenotypes. Results are computed as an ensemble across 11 batches of the test set. The colored markers for each cell phenotype denote the median of the distribution of scores across the 11 batches. The dashed black line denotes the mean of all individual class medians (MoM(PR-AUC)).

Chapter 5

Discussion and Future Outlook

This chapter provides a discussion of the results shown in Chapter 4. Furthermore, whenever applicable, recommendations are offered on how the work in this thesis could be further developed in future research.

5-1 Comparison of `TNBC` Encoder to Publicly Available Pre-trained Encoders

In the case of classifying entire tumor sections, I have taken a similar approach as Ciga et al. [16] and included a pre-trained encoder in my comparison that was pre-trained on ImageNet [76] and published as part of the original study presenting the Simple Framework for Contrastive Learning of Visual Representations (SimCLR) [14] (here referred to as the `SIMCLR` encoder).

ImageNet pre-training is generally considered state of the art for initializing encoder networks and transferring to other vision tasks, but this is not guaranteed to ensure good results in all image domains or tasks [77]. Similar to the findings by Ciga et al., I observed that the encoder network pre-trained on ImageNet, despite providing generally useful features for tumor classification, was easily outperformed by an encoder network pre-trained on images directly related to the task at hand (in this case, the `TNBC` encoder which was pre-trained on triple-negative breast cancer (TNBC) images). In fact, the median classification accuracy for the `SIMCLR` encoder even decreased when its encoded feature representations for all channels of a multiplex immunofluorescence (MxIF) image were considered at once in the Siamese configuration. Conversely, the median classification accuracy for the `TNBC` and `CIGA` encoders increased in this case.

The `CIGA` encoder was pre-trained on bright-field microscopy images of tumor tissue from various organs. Arguably, the domain of bright-field microscopy images is closer to the domain of MxIF images than it is to the domain of natural images, i.e., photographs as they appear in ImageNet. Both image domains capture the same underlying subject — biological tissue

— and you could argue that their main difference is simply due to the imaging modality. One very obvious difference is the fact that, for bright-field images, the background consists of bright pixels and the signal of dark pixels, whereas the opposite is the case in fluorescence images. In order to test if the performance of the `CIGA` encoder could transfer better to MxIF images if they only were visually more similar, I color-inverted all images I provided to the encoder, in a scenario I refer to as `CIGA_INV`. Interestingly, for (`CIGA_INV`), the median classification accuracy increase was the highest in the Siamese configuration, whereas the performance in the Single configuration was relatively poor.

I believe that this observation warrants further investigation. Can we pre-train a more general-purpose encoder that learns useful feature representations from both bright-field as well as MxIF images? One key ingredient to this could be a color-inversion augmentation during the pre-training, making the learned features invariant to such color differences and instead focusing on capturing the underlying structural and morphological patterns present in the images.

5-1-1 Regarding Resource Requirements for Training

It is worth noting that the `SIMCLR` reference encoder is a ResNet-50 (Residual Network) network that has been trained with an incomparably large amount of computational resources. In contrast, the encoder network by Ciga et al. (`CIGA`) [16] as well as the encoder network which I trained on the TNBC dataset (`TNBC`) are ResNet-18 networks. Those have been trained for approximately one day using a high-end consumer grade graphics processing unit (GPU) [1]. The fact that relatively high classification accuracy could be achieved with relatively small resources is promising, since we may be able to achieve even better results with more resources and improved algorithms.

5-2 Importance of Spatial Features in Tumor Classification

A notable result of the tumor classifier training in the single configuration shown in Section 4-1-1 is the fact that the `TNBC` encoder was able to produce particularly useful features from the DAPI channel (i.e., containing cell nuclei) for the classification task. In fact, the classifier that was trained on these feature representations outperformed the classifiers that were trained on feature representations generated from the CD3 and CD8 image channels — cell phenotypes whose presence and absence were associated with the identified tumor phenotypes in the study by Hammerl et al. that originally analyzed these images [4].

Hammerl et al. did study the spatial aspect of cell distributions and densities in these tumors (which is why they are referred to as *spatial* tumor phenotypes) and used the `"CD8+"` (and `"CD3+"`) cells to identify the spatial phenotypes, depending on whether they occur in the tumor border, tumor center, or not at all. Figure 5-1 shows representative examples of these

[1] In my case, an NVIDIA RTX3090. Ciga et al. have published the training time and hardware used for their model, allowing me to train my own network accordingly for a fair comparison. Note that SimCLR is a Google Research publication. The authors are not explicitly stating their numbers, but I estimate that the pre-trained encoder of theirs received about $\times 250$ compute compared to the encoders trained by Ciga et al. and me.

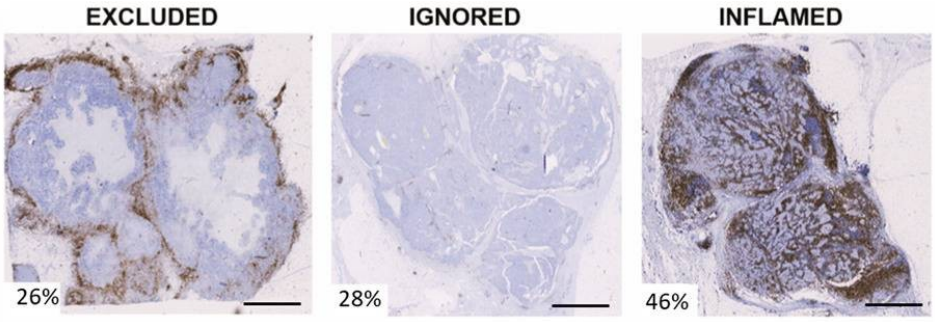


Figure 5-1: Representative stained bright-field microscopy images of the spatial tumor phenotypes "excluded", "ignored", and "inflamed". The percentages indicate the amount of cells labeled `"CD8+"`, which are targeted by the darker colored stain. Scale bars correspond to 5 mm. Image from [4].

spatial tumor phenotypes as bright-field images with an immunohistochemistry (IHC) stain targeting `"CD8+"` cells. Given the highlighting of `"CD8+"` cells, we can clearly see a visual pattern that identifies these tumor phenotypes. The feature representations from the CD8 or CD3 channels of the MxIF images may conceivably contain this information, resulting in good performance for classifiers that were trained on these feature representations. Indeed, all encoder networks were able to embed this information in their feature representations for at least either the CD8 or CD3 channels, leading to the corresponding classifiers achieving relatively high accuracy.

The `TNBC` encoder's ability to generate useful feature representations from the DAPI image channel, which led to higher classification accuracy, indicates that it not only learned a simple "yes/no" representation based on cell detection but also captured spatial features that are indicative of inflammation. This highlights the predictive capabilities of spatial features in tumor microenvironments (TMEs) (as is suggested by ongoing research [3, 5, 4, 7]), even in the absence of specific immune cell phenotype markers.

This observation, however, also has broader implications for the analysis of medical image datasets using self-supervised models: Since DAPI is a widely used stain in fluorescence microscopy, and almost every MxIF dataset includes images with DAPI channels, there are vast amounts of data available that potentially contain valuable spatial information that a self-supervised encoder could use to learn meaningful feature representations. By focusing on extracting and incorporating spatial features from this widely available image data, future research could potentially create more powerful and generalizable classification models applicable to a broader range of MxIF datasets and medical image analysis tasks.

5-3 Pre-trained Encoders and the Siamese Architecture

It should be noted that my proposed Siamese architecture (see Figure 3-6 and Figure 3-8) is not particularly novel on its own, but simply an application of standard neural network design ideas. It is not uncommon to concatenate, or otherwise combine, outputs of several parts of one or several neural networks and feed them into a different part. However, the important implication of using this architecture is that it allows us to **pre-train a single**

encoder network on all image channels of a MxIF image individually. The intuition behind this is that every channel of a MxIF image might as well be considered a separate image that we can extract features from — independently of all its other channels. As a result, the encoder can effectively pre-train on images with an arbitrary amount of channels, potentially making it viable to process modalities such as multiplexed ion beam imaging by time of flight (MIBI-TOF) [78].

By providing the encoder with several image channels and concatenating its feature vector outputs generated from each input, we obtain a representation of a multi-channel image. A disadvantage of using this concatenated feature vector as input to a linear classifier is the loss of flexibility in the number of channels: the input size of the linear classifier must be pre-defined, limiting it to processing feature vectors of a fixed size. However, this limitation could potentially be addressed by employing more advanced architectures, such as transformers [79], which can adapt to varying input sizes and potentially learn complex relationships between channels that a linear (or nonlinear) classifier cannot.

Furthermore, it is worth noting that the feature representations generated by the pre-trained encoder can be utilized for purposes beyond linear classification. They could also be employed for unsupervised methods such as clustering or dimensionality reduction, opening up a range of potential applications in biomedical image analysis.

A specific application that I would propose to investigate is to use a pre-trained encoder, trained according to the methods outlined in this thesis, as a drop-in replacement for the patch-contrastive learning (PCL) module in the NaroNet architecture [18]. NaroNet has proven to be effective for the analysis of complex spatial interactions within the TME. However, its PCL module is limited to self-supervised training on images with a fixed amount of color channels. An encoder network pre-trained on individual image channels could provide NaroNet with a flexibility in image channels while leveraging its powerful architecture. Another recommended application would be the integration into the TME-Analyzer, which I outline in Section 5-9.

5-4 Evaluating Individual Cell Classification and Hyperparameter Optimization

For the classification of individual cells, I could no longer rely on reference encoder networks to compare performance against, since the domain of small image patches centered around individual cells has not been widely considered yet in the application of self-supervised learning (and SimCLR in particular). I therefore decided to simply compare performance internally against my own networks as I optimized their hyperparameters (effectively benchmarking against the labels generated by TME-Analyzer [7]). A useful metric for this was the Area Under the Precision-Recall Curve (PR-AUC), since it is independent of decision thresholds that influence a classifier's performance more strongly if the class distribution of the dataset is imbalanced. This approach also allowed me to investigate the effect of various augmentations used during the pre-training stage, which has not studied as thoroughly in related literature.

5-4-1 Impact of Augmentations in Individual Cell Classification

Although Ciga et al. [16] investigated the effects of augmentations, their approach focused on classifying larger scale tumor images and not on the individual cell level. Furthermore, they analyzed bright-field microscopy images with 3 color channels — an image modality that may require an entirely different set of augmentations compared to MxIF microscopy images. For NaroNet [18], which contains a network that performs contrastive learning on small patches extracted from MxIF microscopy images, the authors did not investigate the effects of augmentations. Furthermore, the image patches used in NaroNet are relatively small with sizes between 10×10 and 15×15 pixels. Whereas NaroNet learns spatial relationships by feeding the learned feature representations into a graph neural network, I decided to capture these spatial features by choosing an image patch size (64×64 pixels) that would include a particular cell and its immediate neighbors. Again, this difference in approach likely facilitates the need for different image augmentations, which is why I did not rely on NaroNet for design decisions of my network.

Choice of Augmentations

Introducing an augmentation in SimCLR teaches the encoder network to make its feature output invariant to that augmentation. The augmentations I have chosen to investigate are somewhat related to microscopy and cell biology:

- **Gaussian blur:** Can help offset blurring due to various factors, such as optical aberrations, imperfections in lenses, or limitations in the resolution of the imaging system.
- **Translation:** Not directly related to microscopy, but can help offset inaccuracies in cell positions obtained from label data.
- **Zoom:** Can help offset slight changes in magnification as well as create robustness against varying size of cells that otherwise have the same phenotype.
- **Brightness/contrast:** Can help offset overexposure or underexposure due to factors such as photobleaching of fluorophores, inconsistent fluorophore concentration during staining, differences in tissue autofluorescence across patients, differences in microscopy-/imaging equipment used, etc. all of which have an effect on brightness and contrast levels.

The augmentations that resulted in best performance may appear relatively "mundane" at first sight: a moderate zoom out of up to $\times 2$ and moderate contrast/brightness adjustment of up to 50 %. The level of contrast and brightness adjustment are lower compared to the 80 % found to be optimal by Ciga et al. [16] for bright-field microscopy images. Interestingly, 80 % for contrast and brightness adjustment were also proposed in the original SimCLR paper [14].

Effects of Zoom

The moderate zoom out of up to $\times 2$ causes the encoder network to produce similar feature outputs for image patches containing cells of different sizes. The center of the image patch

always remains the same and merely becomes slightly smaller when the zoom out causes the network to observe more of the center's surroundings. This suggests that the encoder puts high importance on whatever object or pattern is located at the center of the image and learns to produce similar feature outputs for this object or pattern appearing in different surroundings, which results in better cell classification performance when a classifier uses these features as input. I can propose a few interesting ideas for augmentations that could be applied to investigate this further:

- Cut out the center of an image patch: Depending on whether this improves performance or not, we can get a better understanding of how important the image center is for learning cell features.
- Cut out anything but the center: Depending on whether this improves performance or not, we can get a better understanding of how important the cell's surroundings are for learning cell features.
- Cut out the center of an image patch and "transplant" it into a different image patch: Depending on whether this improves performance or not, we can get a better understanding of how important the spatial relationship between a cell and its surroundings is.

Effects of Translation

The focus on the image patch center may also explain why the translation augmentation resulted in a decrease of performance for even the smallest shift, as shown in Section 4-2-2. The performance decreased very strongly beyond translation shifts of 15 pixels. According to the label data obtained by TME-Analyzer [7], the average cell diameter in the dataset is roughly 30 pixels. Introducing a shift of 15 or more pixels means that the cell that the image patch is centered around is most likely no longer part of the center at all.

For this reason, I tested the special translation augmentation that simply shifts both image patches of the augmentation pair by the same amount and in the same direction. While this would eventually result in an image patch no longer being centered on a cell, it turns out that this type of augmentation had virtually no impact on the quality of feature outputs. This suggests that, while the encoder has a strong focus on whatever is located at the center of an image patch, it does not matter whether it is the actual center of a cell, the edge of a cell or simply empty space.

The last assumption is supported by the fact that even a shift to a completely random location within the tissue image had no effect on performance. This result is significant, because it suggests that **we do not need to know the cell locations for pre-training an encoder network that is supposed to improve cell classification performance**.

If randomly sampled image patches are enough, a lot of time can be saved trying to determine cell locations before beginning to pre-train the encoder. When using the pre-trained encoder to generate feature representations for cells, however, we will need to know cell locations. There are several methods to obtain cell locations via segmentation in MxIF images, particularly when a DAPI channel is available, which highlights cell nuclei. For example: the watershed

algorithm [69], StarDist [70] (both of which are featured in TME-Analyzer) or a powerful general-purpose segmentation network such as Segment Anything [80].

It may also be possible to use the pre-trained encoder without requiring cell location data at all. When dividing a full tissue image into (overlapping) patches at regular intervals, an encoder could compute feature representations for each of these patches. This would result in a lower resolution "heatmap" of the original image, which could be analyzed to find interesting patterns and features in the tumor.

Interesting Augmentations for Future Research

For future research, it may be interesting to consider more augmentations. In a recent paper by Toth et al. [81], the authors could demonstrate improved cell classification performance of a supervised network by introducing a fisheye transformation, which they argue is due to the fact that the transformed images contain more of the cell's "surrounding microenvironment". This fisheye transformation, as well as other biologically/microscopy inspired augmentations could be explored further:

- **Elastic deformation:** Applying elastic deformations to images can simulate variations in cell shape and morphology, which could be beneficial in learning more generalizable features for cell classification.
- **Noise injection:** Adding different types of noise to the images can simulate the effects of imaging noise, helping the network become more robust to real-world variations in image quality.

5-4-2 Impact of Loss Functions and Sampling in Individual Cell Classification

As shown in Section 4-2-1, the choice of loss function and data sampling turned out to have a stronger impact on performance than the augmentations. The thorough investigation was motivated by the fact that, due to MxIF images not containing the same amount of information in every channel, many of the extracted image patches from the TNBC dataset were very dark and contained little or no information. This can cause an issue when sampling from the dataset and applying the normalized temperature cross-entropy (NT-Xent) loss function to the computed feature projections.

Consider the example minibatch of 8 image patch augmentation pairs shown in Figure 5-2: The image pairs $(3, 4)$, $(5, 6)$ and $(13, 14)$ may be considered mostly void of information. The computed feature projections of all images of these pairs will consequently be very similar. The alignment loss term in NT-Xent will be low since the cosine similarity within each image pair is high. However, the distribution loss term will also be low when computed across these three pairs, since the cosine similarity across each image pair is also high. This could lead to the encoder network not being able to learn properly, since it receives conflicting information during pre-training ("these 2 projections of identical images should be made as similar as possible, whereas these 2 projections of identical images should be made as dissimilar as possible").

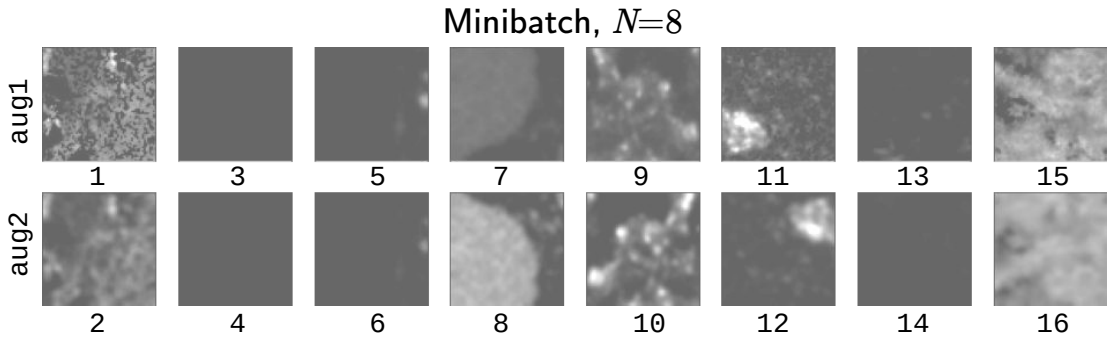


Figure 5-2: Minibatch with $N = 8$ resulting in 8 augmented image pairs for a total of 16 images with indices as shown. In the TNBC image set, many image patches are dark and contain little to no information. This can cause a contrastive loss function to fail learning properly by receiving conflicting information during training. For better visibility, images are displayed with a brightness adjustment of 90 % and contrast adjustment of 50 %.

Although the temperature parameter τ of the NT-Xent loss function can help to mitigate this type of problem to some degree, I decided to try out more ideas in the form of modifying the loss function or data sampling strategy. While rejection sampling of images that fell below a certain brightness threshold seemed promising, it turned out to be computationally expensive during training and did not produce good results. This is most likely because black images make up a significant amount of the dataset and, as a result, the encoder network should get an appropriate amount of opportunities to learn features for these images during pre-training. The use of rejection sampling artificially lowers the relative amount of black images in the dataset to $1/N$ where N is the chosen batch size. The degraded performance of contrastive learning methods when dataset diversity is low is a known problem, which methods such as prototypical contrastive learning [82] aim to address.

As a simple alternative, I tested using only the alignment loss term of the NT-Xent loss function. While this did not result in good performance, future work could investigate more sophisticated non-contrastive methods, such as Barlow Twins [83] and VICReg [84].

The strategy of "black image marking" turned out to result in best performance when combined with an optimal τ parameter. The modification I made to the NT-Xent loss function essentially results in it only selectively applying the distribution loss term based on an arbitrary criterion. In this case, I chose the relatively simple criterion of the intensity of the brightest pixel in the image patch needing to be below a certain threshold. While this introduced another hyperparameter to optimize for, it also improved the learned feature representations of the encoder. Using this modified loss function, there are certainly more sophisticated marking criteria that could be investigated in future work such as thresholds based on the mean and standard deviation of pixel intensities within an image patch.

5-5 Exploring Dataset Diversity and Size for Improved Pre-training

An idea to improve dataset diversity was to include images from datasets other than the TNBC dataset in the pre-training process. This was inspired by the CellPose dataset [73], a

dataset for training and improving cell segmentation algorithms, which contains various unusual images (see Figure 3-3) aside from microscopy images of tissue and cells. Consequently, I chose to include CellPose to investigate whether it could accomplish the same in the context of self-supervised pre-training of an encoder network.

The second dataset I chose to include for this purpose was the STL-10 dataset [74]. Since the STL-10 dataset itself is based on the ImageNet dataset [76], including it in the pre-training might provide the same benefit that ImageNet provides for the pre-training of encoders for larger size images. STL-10 was a convenient choice for this, since its small image format of 96×96 pixels loosely centered on an object of interest meant that it would require little pre-processing and could potentially reinforce the same learning patterns that are required to learn good features for individual cells in the TNBC dataset.

While the inclusion of these datasets did not lead to improved performance in this thesis, it may be an approach worth investigating further in future work. It is likely that image data from foreign image domains can improve learning of better feature representations, but this might require re-evaluation of the augmentations used.

Part of the work by Ciga et al. [16] has focused on gathering datasets of bright-field microscopy images to include during the pre-training process of their encoder network. The TNBC dataset itself was more than large enough for a network that could be feasibly trained within this thesis. Furthermore, Erasmus University Medical Center could potentially provide more MxIF images to analyze. The result that increasing the dataset size during pre-training improves the performance of a downstream task such as classification (see Section 4-2-3) is in line with Ciga et al.'s findings (as well as self-supervised learning literature in general) and gives a promising outlook for future work where larger networks could be trained and provide even better performance.

Examples of images that could possibly be included during pre-training are:

- Images of other tumors,
- bright-field microscopy images (as well as phase-contrast microscopy, etc.), including a color-inversion augmentation (as mentioned in Section 5-1),
- MIBI-TOF [78] images (as mentioned in Section 5-3) and images from other high-plex modalities such as imaging mass cytometry (IMC) and Cyclic Immunofluorescence (CycIF) [85].

5-6 Addressing Label Scarcity

The optimizations in pre-training leading to more label-efficiency in the downstream classification task (Section 4-3-1) strongly indicate that self-supervised learning methods could indeed be used to address the issue of label scarcity in medical imaging data. The cells most heavily affected by this investigation were those with rare phenotype labels such as `"CD56+"` and `"CD68+"`, whereas the amount of labels used was already very low for the remaining phenotypes (Table 3-5).

For example, less than 0.9 % of `"CK+"` labels were used for the supervised training of the classifiers, with performance for these cells being consistently high. For the cells that are

currently undersampled in the training set, it is worth investigating how much higher the classification performance can increase if we add more labels. Doing this would also give us a better understanding of the encoder's "performance ceiling" (which we currently only know for "CD56+") and would help us investigate which augmentations could add even more performance.

5-7 High Variance in Classification Performance for Rare Cells

A notable effect that can be observed in every violin plot shown in Section 4-2 and Section 4-3 is that variance for PR-AUC("CD56+") and, to a lesser extent, PR-AUC("CD20+"), is relatively high. This can be partly explained by the fact that both "CD56+" and "CD20+" are rare labels. However, note that "CD68+", which is less common than "CD20+", is not as heavily affected.

Although I balanced the training set such that every label occurs with similar frequency, the dataset that was provided during the self-supervised pre-training was sampled randomly. As a result, the encoder networks encountered these rare cells infrequently during pre-training, causing them to not learn feature representations as well for rare cells as opposed to more common cells. In the downstream task, this can lead to more uncertainty.

Furthermore, it must be stressed that the TME-Analyzer labels, despite shown to be predictive for patient outcomes [7], are not ground truth and could also include inaccuracies caused by certain cell phenotypes being less common and therefore more difficult to label accurately. This can have a compounding effect when we use these labels to train classifiers.

5-8 Practical Application: Leveraging Higher-level Information for Efficient Annotation

To give an example of how the resulting network of this thesis could be used, I constructed an example where I performed supervised training of a classifier on cells exclusively from one of the spatial phenotypes (see Figure 5-1) and tested those classifiers on cells from the other spatial phenotypes (Section 4-3-2). In this example, I showed that a classifier trained on cells from tumors of the excluded phenotype gave the best performance when tested on cells from the inflamed and ignored phenotypes. In other words, cells from the excluded tumor phenotypes lead to the best generalization performance.

This type of result can be used when making a decision on where the labor of annotating medical images should be allocated in order to make the most of the limited amount of man hours available[2]. This is an example of how higher-level information, such as broad knowledge of a tumor phenotype, can guide us to better determine lower-level information, such as cell phenotypes.

[2]Ideally, we would simply be able to order more cells with a rare phenotype, such as CD56, to be labeled in order to be able to increase classification performance on those cells. This, however, would require an algorithm that can reliably detect those cells in the first place...

5-9 Potential Integration into TME-Analyzer

The TME-Analyzer tool, developed by Balcioglu et al. [7], was used to generate the label data for the TNBC dataset. It is a semi-automatic analysis tool that can be used to perform individual cell classification with the following workflow:

1. Load a MxIF image of a tumor section
2. Segment cell nuclei using StarDist [70]
3. Perform Voronoi expansion of nuclei segmentation masks to approximate cell segmentation masks
4. Perform interactive thresholding using various metrics to determine cell phenotypes

An encoder network pre-trained on MxIF images of tumor tissue could be integrated into TME-Analyzer and assist the user during the workflow. Upon loading the image, the pre-trained encoder could compute a feature embedding of the entire image, which, together with the image, could serve as input to a modern segmentation algorithm such as Segment Anything [80] to improve cell segmentation based on the detected tumor phenotype. With the cells segmented, their respective image patches could be fed into the pre-trained encoder in order to compute feature embeddings for every detected cell or other regions of interest. These feature embeddings could be integrated into the thresholding step to assist the user in determining cell phenotypes or perform clustering analysis to discover new regions of interest. This human-in-the-loop approach may even result in discovery of novel features and phenotypes that could be further investigated [86].

Chapter 6

Conclusion

In this thesis, as part of a collaboration between TU Delft and Erasmus University Medical Center (Erasmus MC), I explored the potential of self-supervised learning algorithms, specifically SimCLR (Simple Framework for Contrastive Learning of Visual Representations), to extract meaningful feature representations from multiplex immunofluorescence (MxIF) microscopy images of triple-negative breast cancer (TNBC) tissue.

The primary motivation for this investigation was to enhance the capabilities of the TME-Analyzer, a tool developed by Erasmus MC for the analysis of tumor microenvironments (TMEs), which currently relies on manual annotation of image data. The widespread issue of label scarcity in medical imaging, along with the promising ability of self-supervised learning algorithms to mitigate this challenge, inspired this research to improve the efficiency and automation of the TME-Analyzer's workflow.

In light of the images in the TNBC dataset including 8 color channels, this study aimed to address the less explored domain of MxIF images and adapt the SimCLR model accordingly, ensuring its flexibility in handling varying numbers of image channels and learning feature representations that are useful for improving the label-efficiency in the downstream task of cell classification.

Research Question 1: How can a self-supervised learning model be designed to learn features from MxIF images with an arbitrary number of color channels?

MxIF images can be split into their individual image channels, which can be supplied to the SimCLR encoder individually during pre-training. As a result, the encoder can learn feature representations from images with an arbitrary amount of color channels. In order to use these features to perform classification of MxIF images, I explored two configurations:

1. Single configuration: Feed the 8 individual channel's feature representations into 8 separate classifiers.

2. Siamese configuration: Feed the concatenation of all 8 channel feature representations into one classifier.

In an experiment of classifying entire tumor sections in the TNBC dataset, I showed that the Siamese configuration achieved a median classification accuracy of 0.937, compared to the Single configuration where the best performing classifier (using the DAPI channel) achieved a median classification accuracy 0.913, marking a +2.63 % increase. Furthermore, encoder networks that were not pre-trained on the TNBC dataset resulted in worse classification performance.

Research Question 2: What are the hyperparameters for such a model that improve feature learning of individual cells in MxIF images?

To answer this question, I performed a hyperparameter search, which I divided into 4 rounds:

1. Determine best loss function and data sampling strategy during pre-training.

2. Determine the best augmentations to use during pre-training.

3. Investigate the effects of varying dataset size and data sources during pre-training.

4. Investigate the effects of changing the amount of labeled data available during the supervised classifier training (this round addresses the next question).

Following the previous results, I employed the Siamese configuration and performed experiments of classifying individual cells in the TNBC dataset. Every hyperparameter change resulted in a different encoder/classifier pair to be evaluated. As a performance metric, I used the classifier's Area Under the Precision-Recall Curve (PR-AUC). I benchmarked the performance of every network internally against each other, with the network that achieved the highest classifier PR-AUC indicating the best performance and therefore marking the best hyperparameters.

The best hyperparameters found were:

1. Use the normalized temperature cross-entropy (NT-Xent) loss function with temperature parameter $\tau = 0.05$ and a modification to the loss function that excludes images from the distribution loss if their brightest pixel has an intensity of 30 or lower.

2. Apply rotation and flipping with 50 % probability, apply a zoom out with a random zoom factor up to 2×, and adjust the contrast and brightness with random intensities up to ±50 % .

3. Pre-train the encoder on 4 096 000 image patches (maximum amount tested) from the TNBC dataset.

Research Question 3: Can the features learned by such a model improve label-efficiency for the task of individual cell classification?

The answer to this question follows from the investigation performed in round 4 as specified above. Using the pre-trained encoder with the best hyperparameters obtained after round 3, I trained multiple classifiers on top of it and varied the amount of labeled data available.

I found that I could decrease the amount of labels available to 40 % before a noticeable decrease in classifier PR-AUC occurred. Furthermore, the best encoder from round 3 with a classifier trained on only 10 % of label data outperformed the best encoder from round 1 with a classifier trained on 100 % label data. The best encoder from round 3 with a classifier trained on only 30 % of label data outperformed the best encoder from round 2 with a classifier trained on 100 % label data.

The results show that the features learned by the encoder in round 3, which has the best hyperparameters, are of high enough quality that only a fraction of labeled data available is required to match classification performance of encoders from earlier rounds. This demonstrates that the optimized self-supervised learning model can indeed improve label-efficiency for individual cell classification, reducing the reliance on labeled data without sacrificing performance.

Future Outlook

The promising results of using SimCLR for feature learning of tumor sections and individual cells serve as a basis for future research.

During pre-training on tissue section images, the encoder networks learned spatial features from the DAPI channel that were useful for tissue classification. This can be investigated further by leveraging the large amount of DAPI channel images publicly available. Furthermore, the inclusion of images from other image modalities such as bright-field microscopy or other high-plex imaging techniques may improve the quality of learned features.

The hyperparameter search, particularly with respect to augmentations, can be extended in future research. Applying augmentations that are relevant to the microscopy domain and address specific challenges in cell imaging could further improve the performance and generalization of the learned features.

The SimCLR encoders trained as described in this thesis can be used for more tasks than image classification. Possible applications to consider in the future are:

- Enhancing TME-Analyzer: The pre-trained encoder could be incorporated into the TME-Analyzer workflow, potentially improving cell segmentation and assisting users in determining cell phenotypes or performing clustering analysis to discover new regions of interest.
- Integration into NaroNet: The pre-trained encoder could be used as a drop-in replacement for the patch-contrastive learning (PCL) module, providing flexibility in the number of image channels and leveraging the powerful architecture of NaroNet for the analysis of complex spatial interactions within the TME.

In conclusion, this thesis demonstrates the potential of self-supervised learning algorithms, specifically SimCLR, in extracting biologically relevant features from multiplexed immunofluorescence images and improving label-efficiency for tasks like individual cell classification in the context of triple-negative breast cancer. The results lay a foundation for future work in refining self-supervised encoder networks, exploring additional data sources, and developing advanced techniques to enhance existing tools such as the TME-Analyzer or NaroNet. Potential directions for future research include the investigation of domain-specific augmentations, leveraging large amounts of DAPI channel images, and exploring diverse and large datasets for pre-training. This work ultimately contributes to the broader goal of enhancing our understanding of tumor microenvironments and has the potential to improve the capabilities of machine learning models for making accurate predictions in clinical settings, aiding in the development of personalized treatments for patients suffering from cancer.

Appendix A

TNBC Image Dataset Specification

The triple-negative breast cancer (TNBC) dataset consists of 1010 fluorescence microscopy images from 63 patients. Each image corresponds to a stamp of size 670 µm × 502 µm at a resolution of 0.5 μm^2/px, resulting in a digital image of size 1340 px × 1004 px. As specified in [7], patients were either determined to have inflamed tissue (25 patients, 349 images) or non-inflamed tissue (38 patients, 661 images). Furthermore, images were either captured at center locations (513 images) or border locations (497 images) of the tumor.

The images were captured by an *Akoya Vectra 3 Imaging System* [87] with the tissue being stained using the *Opal 7* staining kit [25]. Table A-1 specifies the fluorophores used and their targeted markers. During imaging, the *Vectra 3* cycled through combinations of 5 excitation filters and 22 emission filters (see Figure 2-2) for a total of 35 *spectral* images per tissue section. Table A-2 shows the combinations of excitation filters and emission filters used for each of these images. According to the specification of the filters and fluorophores used, the *spectral* images were then unmixed to produce 8 *color channel* images per tissue section: one for every fluorophore and an additional one for tissue autofluorescence (background).

The resulting *spectral* images are of size 1340 × 1004 × 35 with 16-bit color depth and are saved in the proprietary `.im3` file format. The resulting *color channel* images are of size 1340 × 1004 × 8 with 32-bit color depth and are saved in the `.tif` file format. More detailed specifications of both formats are given in the inForm user manual [72].

An example of all 35 spectral image channels for a given tissue section is shown in Figure A-1. Here, the channel numbers run from left to right and top to bottom. Note that the pixel brightness values have been normalized for each channel individually. The brightness of these images is normally much lower and this should only serve to give a qualitative insight. It is worth comparing how the visible tissue changes whenever an excitation filter changes (e.g. between channel 9 and channel 10).

The spectrally unmixed 8 color channel images of the same tissue section are shown in Figure A-2. For reference, the fluorophore and its targeted marker corresponding to the unmixed channel are specified. Note that the pixel brightness values have been normalized for each channel individually.

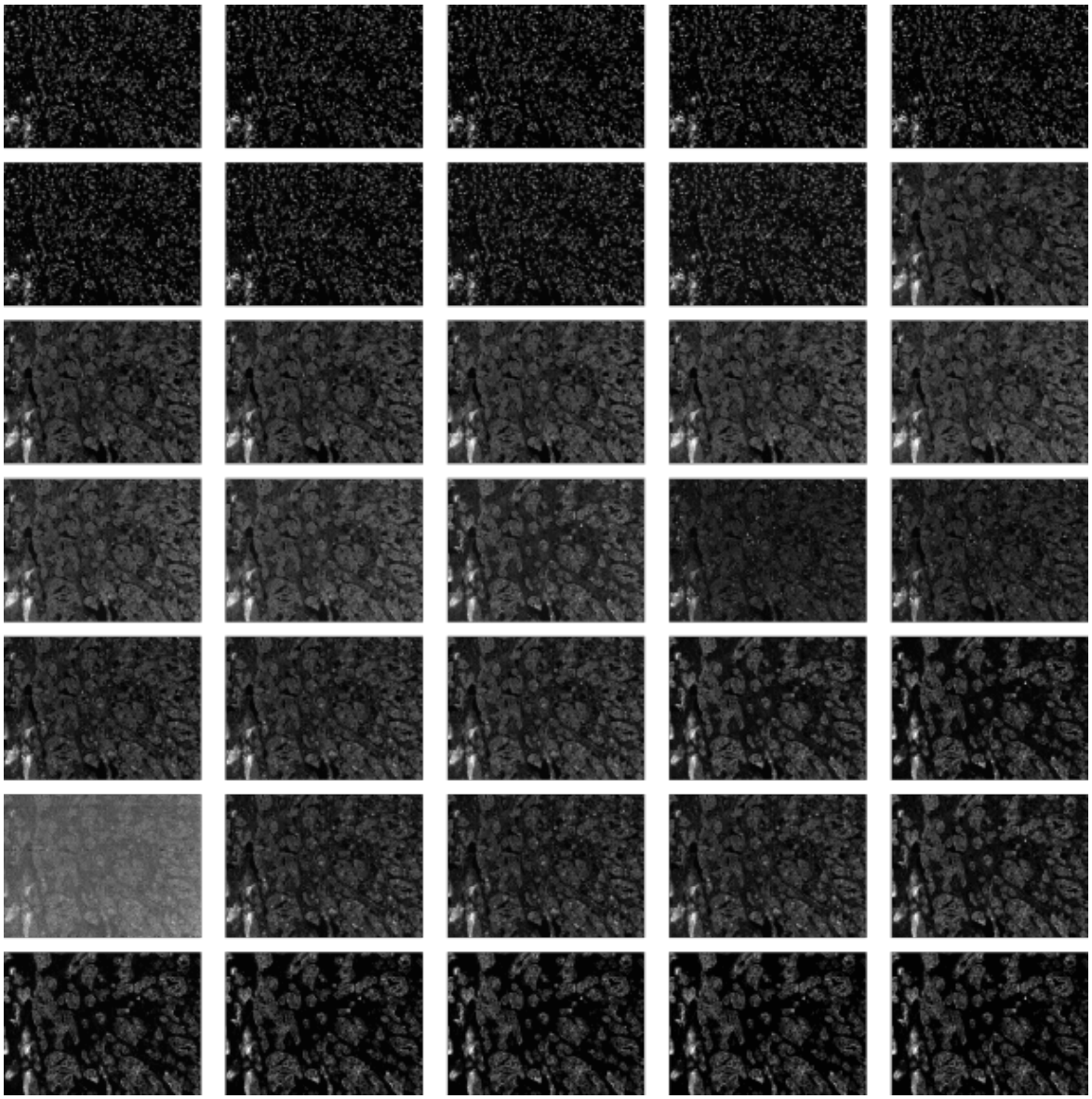
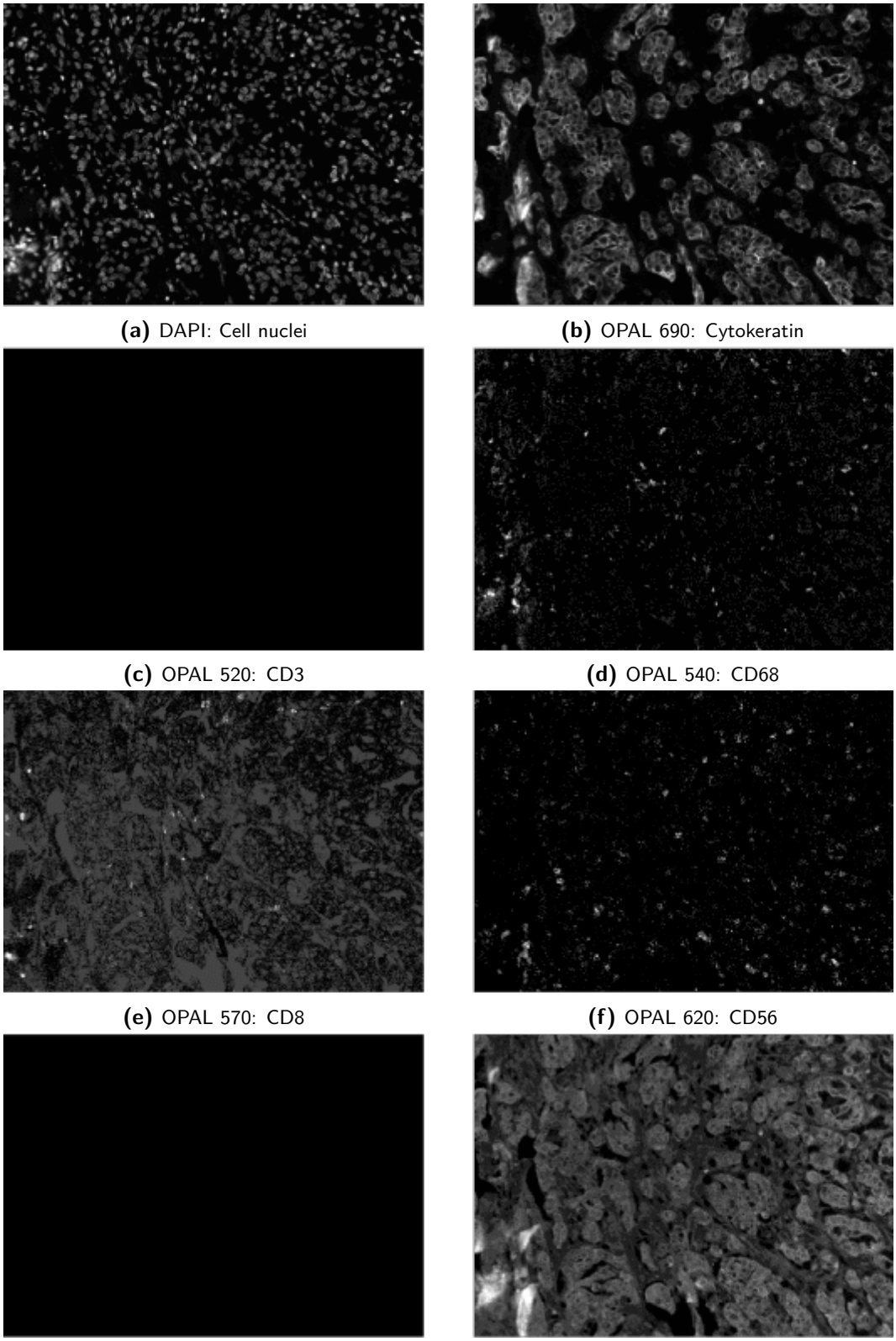


Figure A-1: 35 normalized spectral channels of a tissue sample from the TNBC dataset.



(a) DAPI: Cell nuclei

(b) OPAL 690: Cytokeratin

(c) OPAL 520: CD3

(d) OPAL 540: CD68

(e) OPAL 570: CD8

(f) OPAL 620: CD56

(g) OPAL 650: CD20

(h) Autofluorescence

Figure A-2: 8 normalized color channels of a tissue sample from the TNBC dataset.

Table A-1: Fluorophores and target markers for TNBC images. Some images have the markers for channels 6 and 7 switched.

Color channel #	**Fluorophore**	**Marker**
1	DAPI	Cell nuclei/DNA
3	OPAL520	CD3
4	OPAL540	CD68
5	OPAL570	CD8
6	OPAL620	CD56/CD20
7	OPAL650	CD20/CD56
2	OPAL690	Cytokeratin

Table A-2: Excitation and emission filter combinations for TNBC spectral images. The emission filters are band-pass filters with a range of 20 nm, the quantities in parentheses mark their central wavelength.

Spectral channel #	**Excitation filter λ [nm]**	**Emission filter λ [nm]**
1	410–550	430–450 (440)
2		450–470 (460)
3		470–490 (480)
4		490–510 (500)
5		510–530 (520)
6		530–550 (540)
7		550–570 (560)
8		570–590 (580)
9		590–610 (600)
10	510–650	510–530 (520)
11		530–550 (540)
12		550–570 (560)
13		570–590 (580)
14		590–610 (600)
15		610–630 (620)
16		630–650 (640)
17		650–670 (660)
18		670–690 (680)
19	575–650	560–580 (570)
20		580–600 (590)
21		600–620 (610)
22		620–640 (630)
23		640–660 (650)
24		660–680 (670)
25		680–700 (690)
26	610–750	570–590 (580)
27		590–610 (600)
28		610–630 (620)
29		630–650 (640)
30		650–670 (660)
31		670–690 (680)
32		690–710 (700)
33	660–740	670–690 (680)
34		690–710 (700)
35		710–730 (720)

Appendix B

Refresher on Basic Optics Properties

Figure B-1 shows various objects of interest in the life sciences on a size scale, with the average size of cells found in human and animal tissue in the range between 10 µm and 100 µm. The size of cells, specifically with respect to the indicated *Abbe's diffraction limit*, imposes a requirement on the **magnification** and **resolution** of an optical system. Furthermore, the **contrast** of the imaged cells plays an important role in our ability to analyze the resulting images.

This appendix will give a quick review of these basic optics properties.

B-1 Lenses & Magnification

The thin-lens equation (see also Figure B-2),

$$\frac{1}{f} \approx (n-1)\left(\frac{1}{R_1} - \frac{1}{R_2}\right), \tag{B-1}$$

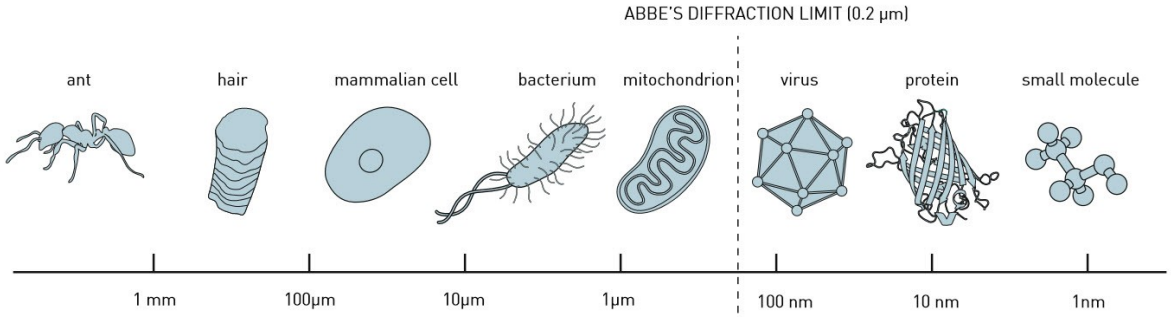


Figure B-1: Various objects of interest in the life sciences on a size scale. The diffraction limit specifies a lower bound on the size of objects that can be observed with conventional optical imaging systems. Image from [88]

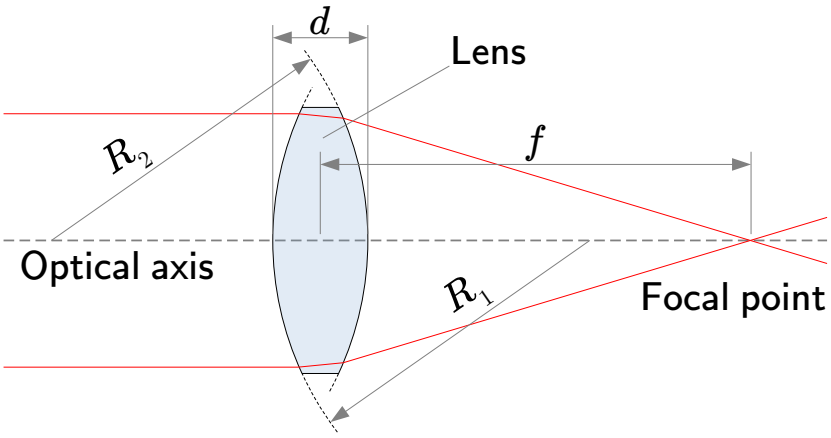


Figure B-2: Geometrical properties of a lens. Negative radii R will result in a concave lens that diverges light, i.e., the focal point f will be located in front of the lens. For the thin-lens equation, the thickness d is assumed to be much smaller than the lens' radii. Image adapted from [90].

relates the lens' focal length f to its shape expressed by the radii of curvature R_1 and R_2 and its material specified by the refractive index n [89]. Common refractive indices are 1.00 for air, 1.33 for water and between 1.50 and 2.00 for typical glass lenses. For thin lenses, the lens thickness d is assumed to be much smaller than R_1 and R_2.

Figure B-3 shows a typical two-lens system configuration. Here, the specimen O is located in the focal plane of the objective lens at distance f_{obj} and projected as image B in the image plane located at distance f_{tl} from the tube lens. Using this lens placement, the magnification M of the system can be expressed as (see [89])

$$M = \frac{B}{O} = \frac{f_{\text{tl}}}{f_{\text{obj}}}. \tag{B-2}$$

Note that the light between the two lenses is collimated, i.e., all light beams are parallel. This is also referred to as infinity-correction and provides the main advantage that the distance between the two lenses does not affect the system's magnification.

Figure B-4 shows a microscope configuration comprising two infinity-corrected magnification steps. The left part of the system is identical to that shown in Figure B-3 and its primary image output serves as input to the right part, consisting of an eyepiece lens and focusing lens (e.g. the observer's eye), which projects a secondary image. The magnification of the right part of the system can be similarly determined by Eq. (B-2) and the total magnification of the system is simply $M_{\text{total}} = M_{\text{left}} \times M_{\text{right}}$. By arranging several lens configurations, high magnifications can be achieved. Although, in theory, magnification can be arbitrarily high, higher magnification does not improve the optical resolution of a system.

B-2 Optical Resolution

So far, we have only considered *geometrical* optics, which models light simply as rays redirected by various optical elements such as lenses. In order to determine an optical system's

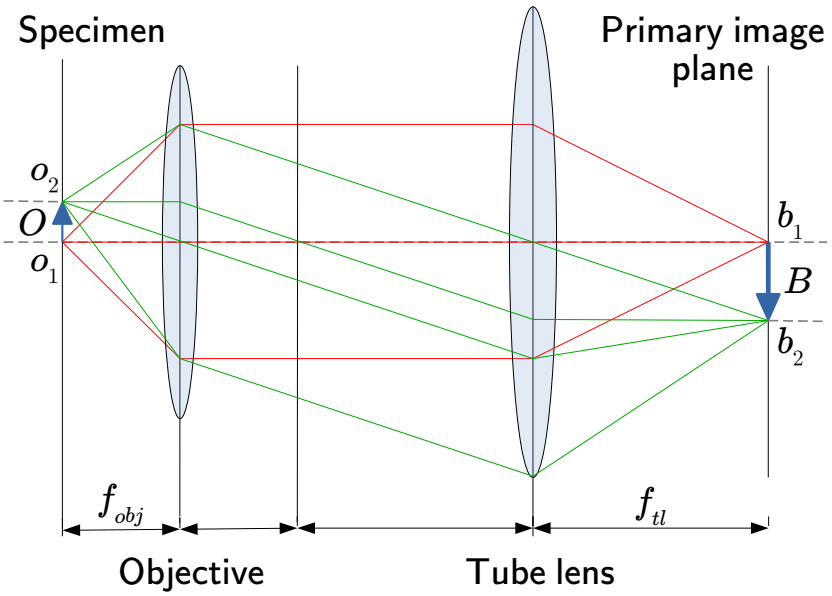


Figure B-3: Infinity-corrected two-lens system. Light is emitted from the object O to form an image B. The red paths show light emitted from the base of the object o_1 and the green paths show light emitted from the top of the object o_2, which arrive at the base b_1 and top b_2 of the image, respectively. Note that all light rays are parallel in-between the objective and tube lens. Image adapted from [91].

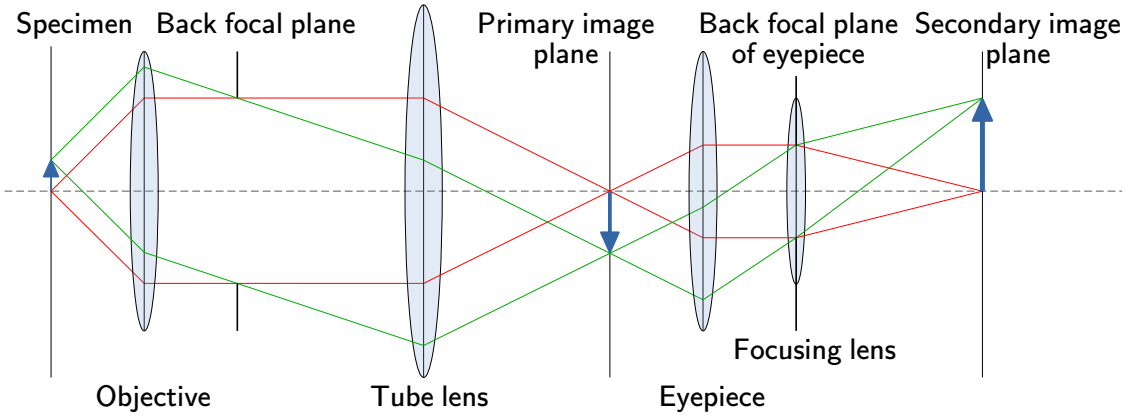
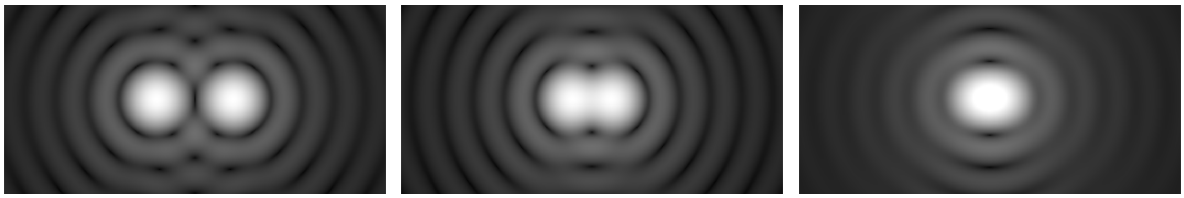


Figure B-4: Two infinity-corrected two-lens systems in series can provide greater magnification. Light paths are simplified compared to Figure B-3. In case of an analog microscope, the focusing lens will be the eye of the observer. Image adapted from [91].



(a) Twice the distance to the first minimum.

(b) Exactly the distance to the first minimum (the Rayleigh criterion).

(c) Half the distance to the first minimum.

Figure B-5: Two airy disks at various spacings. The Rayleigh criterion marks the distance at which projections of two separate point sources can no longer be reliably distinguished from each other. Image from [92].

resolving power, however, we need to consider *wave* optics, which models light as electromagnetic waves. As a result, image formation is partially determined by diffraction of light rather than being simply a point-to-point mapping between object plane and image plane.

An optical system's resolving power is typically expressed as an angular or lateral resolution. In case of lateral resolution, it is the minimum distance between two separate entities such that they are still distinguishable as such. There are many estimates that give a theoretical lower limit to optical resolution, one of which is the Rayleigh resolution criterion [91] (see Section B-4 for a derivation):

$$d_{\mathrm{R}} = \frac{0.61\lambda}{\mathrm{NA}} \quad \text{for incoherent light,} \tag{B-3}$$

$$d_{\mathrm{R}} = \frac{1.22\lambda}{\mathrm{NA}} \quad \text{for coherent light.} \tag{B-4}$$

Here, d_{R} is the minimum distance required between two light point sources in order to be resolved, λ is the wavelength of light considered and $\mathrm{NA} = n\sin(\theta)$ is the numerical aperture, which describes a system's effective angular range over which it can transmit light by relating it to the refractive index n and its half angular aperture θ . For example, consider a fluorescent microscope with a relatively high NA of 1.4 and whose specimen fluoresce green light ($\lambda = 550\,\mathrm{nm}$). Since the fluorescent light emitted by the multiple fluorophores is typically considered incoherent, we can determine the theoretical resolution in this scenario to be about 240 nm.

The Rayleigh resolution criterion assumes that a point source in the object plane is diffracted by the optical system to form an Airy disk pattern in the image plane. This is illustrated in Figure B-5: In Figure B-5a, the two Airy disks projected onto the image plane are at a sufficient distance to still be distinguishable. In Figure B-5b, the two Airy disks are located at a distance specified by the Rayleigh resolution criterion and are just barely distinguishable. In Figure B-5c, the two Airy disks are located too close to each other to be distinguishable and the resulting image may be misinterpreted to have originated from a single point source in the object plane.

Although magnification has, in theory, no impact on resolution as previously stated, there are practical considerations when choosing magnification for a given resolution when capturing images using digital image sensors. Consider the case as in Figure B-5b where the maxima of two Airy disks are located at a distance d_{res} and an individual pixel of the image sensor

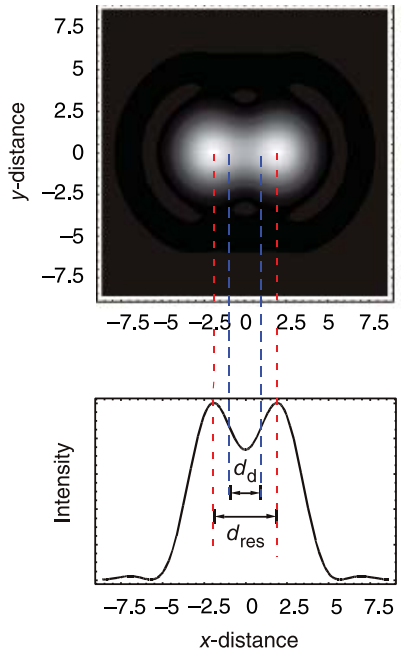


Figure B-6: Two Airy disks located at the Rayleigh resolution criterion d_{res} with unit magnification ($M = 1$). The top shows an image representation and the bottom an intensity histogram when cutting the image at $y = 0$. The sensor's pixel size d_{d} must be chosen small enough depending on the magnification in order to optimally resolve small details. Image from [91].

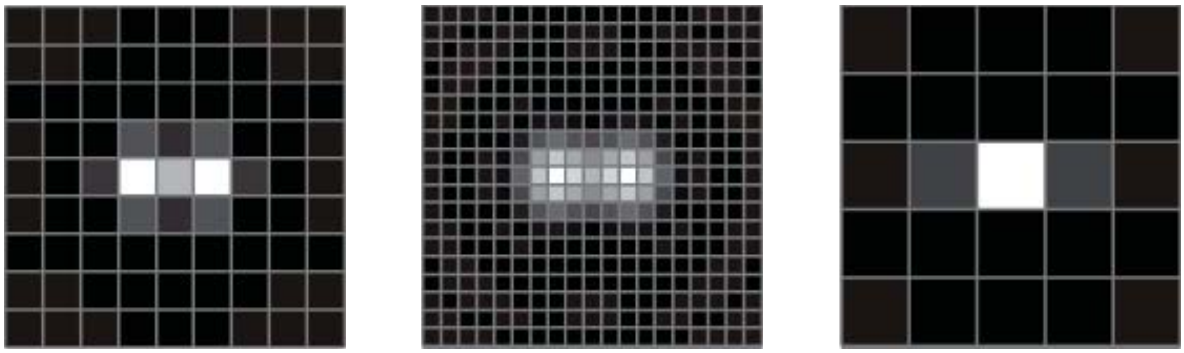
is of size d_{d}. Figure B-6 illustrates this with an additional section view through the center of the Airy disks to show the resulting light intensity on a 2D plot. In order for the two disks to be individually distinguishable in an image of discrete pixels, we require at least one pixel distance between the two pixels corresponding to the disk maxima. This leads to the requirement that the pixel size must not be larger than half the resulting Airy disk diameter d_{Airy}. Since d_{Airy} is dependent on magnification, we can express the condition for optimal magnification as follows [91]:

$$M_{\text{opt}} \approx \frac{2d_{\text{d}}}{d_{\text{R}}}, \tag{B-5}$$

where d_{R} is the Rayleigh resolution distance as specified in Eqs. (B-3) and (B-4). Figure B-7 shows the projected image of Figure B-6 captured by sensors of varying pixel size. Notably, the critical condition $d_{\text{d}} = 0.5Md_{\text{R}}$ specified in Eq. (B-5) can be seen in Figure B-7a, the case of super-resolution $d_{\text{d}} < 0.5Md_{\text{R}}$ in Figure B-7b, and the case of sub-resolution $d_{\text{d}} > 0.5Md_{\text{R}}$ in Figure B-7c. Returning to the example fluorescence microscope with resolution of 240 nm and choosing an image sensor with pixel size of 6 µm × 6 µm, we would require a magnification of at least ×50 in order to optimally capture the image.

B-3 Contrast

Proper magnification and resolution alone are not enough to make an object visible under a microscope, but sufficient contrast is also required in order to be able to tell apart an object



(a) $d_{\mathrm{d}} = 0.5Md_{\mathrm{R}}$, the two points are distinguishable (critical condition).

(b) $d_{\mathrm{d}} = 0.24Md_{\mathrm{R}}$, the two points are distinguishable (super-resolution).

(c) $d_{\mathrm{d}} = 0.9Md_{\mathrm{R}}$, the two points are indistinguishable (sub-resolution).

Figure B-7: The two Airy disks in Figure B-6 projected by an optical system with magnification M and resolution d_{R} as captured by a digital image sensor of varying pixel size d_{d}. Image from [91].

from its surroundings (i.e. distinguish **foreground** from **background**).

A possible mathematical definition for contrast is

$$C = \frac{I_{\mathrm{obj}} - I_{\mathrm{sur}}}{I_{\mathrm{obj}} + I_{\mathrm{sur}}}, \tag{B-6}$$

where I_{obj} and I_{sur} are the light intensities of an object and its surroundings, respectively [91]. Clearly, a large difference between I_{obj} and I_{sur} is desired in order to obtain high contrast.

For the purpose of microscopic imaging, tissue samples are typically sliced very thinly, which results in low contrast according to the ratio in Eq. (B-6). This is necessary in order for light to be able to penetrate the sample and make details visible, so increasing slice thickness is not an option for increasing contrast. In practice, samples are usually stained in order to make the objects of interest (tissue and cells) more easily distinguishable from the background. This staining essentially increases the contrast C as given in Eq. (B-6), which is useful for both human as well as algorithmic analysis of tissue images. The staining methods relevant for this thesis are explained in more detail in Section 2-1-2.

B-4 Rayleigh Resolution Criterion

For incoherent light, the far-field intensity diffraction pattern of a plane wave hitting a circular aperture (for example, a lens) is given by

$$I(\theta) = I_0 \left(\frac{2J_1\big(2\pi a \sin(\theta)/\lambda\big)}{2\pi a \sin(\theta)/\lambda} \right)^2, \tag{B-7}$$

where θ is the projection angle between aperture and image plane, I_0 is the light intensity of the central maximum ($\theta = 0$), a is the aperture's radius, λ is the light's wavelength, and J_1

is the Bessel function of the first kind. The Bessel function of the first kind has its first zero at $J_1(3.8317) = 0$, which results in $I(\theta) = 0$ for $2\pi a \sin(\theta)/\lambda = 3.8317$ or $\sin(\theta) \approx 0.61\lambda/a$.

$$\frac{f}{M}\sin(\theta) = 0.61\frac{\lambda}{n}\frac{f}{Ma}, \tag{B-8}$$

$$\frac{r_{\text{im}}}{M} = \frac{0.61\lambda}{nM\sin(\beta)}, \tag{B-9}$$

$$r_{\text{obj}} = \frac{0.61\lambda}{n\sin(\alpha)}, \tag{B-10}$$

$$r_{\text{obj}} = \frac{0.61\lambda}{\text{NA}}. \tag{B-11}$$

Appendix C

Derivation of Equations for the Backpropagation Algorithm

The derivation in this section closely follows the procedure in [41]. Consider the neural network in Figure C-1: the network has L layers and we consider the connection between the kth neuron in layer $l-1$ and the jth neuron in layer l.

Figure C-2 gives a detailed view of the jth neuron in layer l: *activations* $a_n^{(l-1)}$ of neurons from the previous layer $l-1$ are multiplied with *weights* $w_{jn}^{(l)}$ and summed with a *bias* $b_j^{(l)}$. The output of the jth neuron in layer l is its *activation*

$$a_j^{(l)} = \phi\left(\sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}\right), \tag{C-1}$$

that is, it is the neuron's activation function ϕ applied to the weighted sum of activations from the previous layer and its own bias. For the activations of the entire layer l, we can

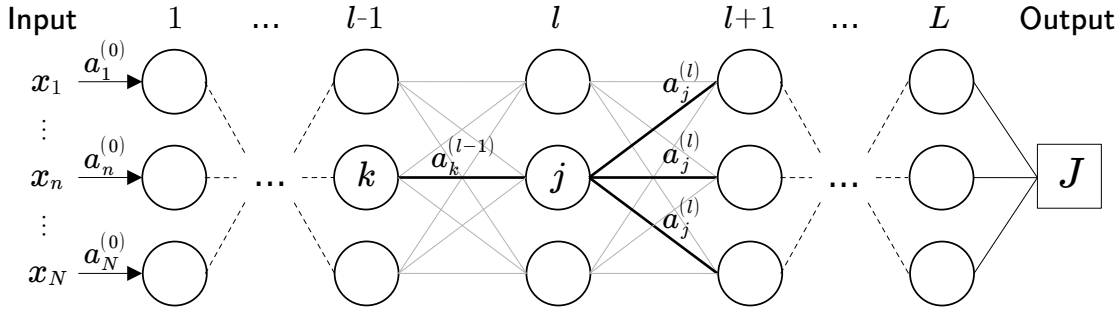


Figure C-1: Schematic overview of a fully-connected neural network. The network has L layers. Highlighted are the connections from the kth neuron in layer $l-1$ to the jth neuron in layer l and its connections to the subsequent layer $l+1$ with their activation terms a specified. Layer 1 presents a special case where the inputs x to the network are treated as the activation of the 0th layer. The outputs from layer L are passed to a cost function J.

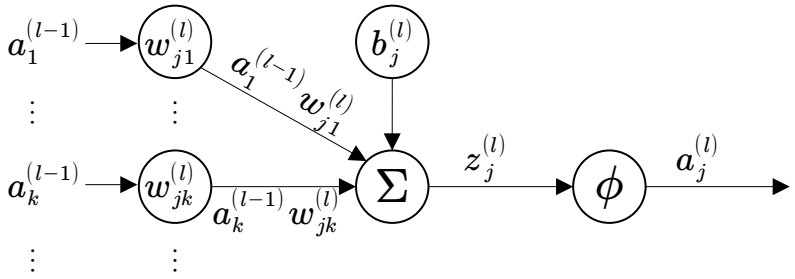


Figure C-2: Schematic detail view of the jth neuron in layer l from Figure C-1 The activations $a_n^{(l-1)}$ from the previous layer are multiplied with weight terms $w_{jn}^{(l)}$ and summed with a bias term $b_j^{(l)}$. The sum of weighted inputs $z_j^{(l)}$ is passed to an activation function ϕ, which produces the neuron's activation $a_j^{(l)}$, i.e., its output. The principle is similar to the Perceptron shown in Figure 2-14.

express this as a vector

$$\mathbf{a}^{(l)} = \phi\left(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right), \tag{C-2}$$

$$\mathbf{a}^{(l)} = \phi\left(\mathbf{z}^{(l)}\right), \tag{C-3}$$

where $\mathbf{z}$ is a vector containing the *weighted inputs* $z_j^{(l)}$ to the jth neurons in layer l. Note that we assume ϕ is applied individually to each element of its input vector.

Furthermore, we introduce an additional intermediate quantity

$$\delta_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}}, \tag{C-4}$$

which relates the cost function J to be optimized to the weighted input of the jth neuron in layer l. $\delta_j^{(l)}$ is the *error* of the jth neuron in layer l. A neuron's error gives an indication as to how strongly a it affects the final cost, whose minimization with respect to every neuron's weight and bias we are interested in. The definition of δ will prove useful in computing this efficiently.

We can now define

$$\delta_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}}\frac{\partial a_j^{(L)}}{\partial a_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}, \tag{C-5}$$

$$= \frac{\partial J}{\partial a_j^{(L)}}\phi'\left(z_j^{(L)}\right), \tag{C-6}$$

where we used the chain rule and the definition of the activation in Eq. (C-3) in order to express the error of the jth neuron at the final layer L in terms of the partial derivative of the cost function with respect to the activation in layer L and the derivative of the activation function itself. Again, we can write Eq. (C-6) in vector notation:

$$\boldsymbol{\delta}^{(L)} = \nabla_a J \circ \phi'\left(\mathbf{z}^{(L)}\right), \tag{C-7}$$

where $\circ$ denotes the element-wise multiplication between two matrices of the same size (Hadamard product), and ∇_a is the gradient with respect to a. For the error at any other layer l, note that the error at layer $l+1$ is $\delta_k^{(l+1)} = \partial J / \partial z_k^{(l+1)}$:

$$\delta_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \sum_k \frac{\partial z_k^{(l+1)}}{\partial z_k^{(l+1)}} = \sum_k \frac{\partial J}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}, \tag{C-8}$$

$$= \sum_k \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}, \tag{C-9}$$

and furthermore

$$z_k^{(l+1)} = \sum_j w_{kj}^{(l+1)} \phi\left(z_j^{(l)}\right) + b_k^{(l+1)} \tag{C-10}$$

with its partial derivative

$$\frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = w_{kj}^{(l+1)} \phi'\left(z_j^{(l)}\right). \tag{C-11}$$

Substituting Eq. (C-11) in Eq. (C-9) results in

$$\delta_j^{(l)} = \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)} \phi'\left(z_j^{(l)}\right) \tag{C-12}$$

which relates the error of the jth node in layer l to the errors of the nodes in layer $l+1$. In vector notation, this becomes

$$\boldsymbol{\delta}^{(l)} = \left(\mathbf{W}^{(l+1)}\right)^{\mathrm{T}} \boldsymbol{\delta}^{(l+1)} \circ \phi'\left(\mathbf{z}^{(l)}\right) \tag{C-13}$$

We can now derive the expression of the derivatives of the cost function with respect to the weights and biases. For the weights, we write

$$\frac{\partial J}{\partial w_{jk}^{(l)}} = \frac{\partial J}{\partial w_{jk}^{(l)}} \frac{\partial z_j^{(l)}}{\partial z_j^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} \tag{C-14}$$

and evaluate the derivative

$$\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \frac{\partial}{\partial w_{jk}^{(l)}} \left(\sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)} \right) = a_k^{(l-1)}, \tag{C-15}$$

giving us the expression

$$\frac{\partial J}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)}. \tag{C-16}$$

For the bias, we simply have

$$\frac{\partial J}{\partial z_j^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \frac{\partial b_j^{(l)}}{\partial b_j^{(l)}} = \frac{\partial J}{\partial b_j^{(l)}} \frac{\partial b_j^{(l)}}{\partial z_j^{(l)}} = \frac{\partial J}{\partial b_j^{(l)}}, \tag{C-17}$$

giving us the expression

$$\frac{\partial J}{\partial b_j^{(l)}} = \delta_j^{(l)}. \tag{C-18}$$

With Eq. (C-7) and Eq. (C-13) we now have expressions relating a neuron's error to known quantities which have been computed in the forward-pass of the neural network. Utilizing these neuron errors, the derivative of the cost function with respect to a neuron's weight can be determined using Eq. (C-16) and with respect to a neuron's bias using Eq. (C-18).

Appendix D

Progression of the State of the Art in Computer Vision

The field of computer vision has experienced tremendous advancements over the years, largely due to the development of innovative neural network architectures and the availability of large-scale datasets for training. This appendix provides an overview of the progression of the state of the art in computer vision, using the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) as a framework to showcase various milestone architectures such as AlexNet, InceptionNet, ResNets, and Vision Transformers. Starting with the foundational MNIST dataset and moving through CIFAR and other large datasets like Microsoft COCO and ImageNet, this appendix is intended to give some insight on how these datasets have helped shape computer vision models and push the boundaries of performance in object recognition tasks.

D-1 Benchmark Datasets for Vision Tasks

In order to quantify the performance of neural networks, particularly novel architectures, it is common to train them on public image datasets and benchmark their performance. One of the oldest and most well-known datasets for this purpose is the Modified National Institute of Standards and Technology (MNIST) handwritten digit database [61, 93] (often simply referred to as MNIST). The dataset features 70 000 grayscale 28×28 pixel images of handwritten digits 0 to 9. Due to its simplicity, even classic Machine Learning (ML) algorithms can achieve good classification performance on the MNIST dataset, although convolutional neural networks (CNNs) are by far the best performers with modern state-of-the-art networks typically achieving error rates below 1 % and the best performance to date achieved by a CNN with an error rate of 0.09 % [94].

Two similarly sized but more complex datasets have been created by the Canadian Institute For Advanced Research (CIFAR). The datasets feature 60 000 color 32×32 pixel images of

everyday objects and animals. CIFAR-10 has images of 10 classes (6000 each) and CIFAR-100 images of 100 classes (600 each).

In order to train algorithms to better perform computer vision tasks on full-sized photographs of everyday objects and situations, rather than tiny image crops, datasets such as Microsoft Common Objects in Context (COCO) [95, 96] and ImageNet [76, 75] have been created. COCO features 328 000 images with 91 classes and ImageNet features almost 15 million images with over 20 000 classes, not all of which are manually reviewed however.

D-2 ImageNet and the CNN Revolution

Since 2010, the ImageNet project runs the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) which requires contestants to classify a manually reviewed subset of images from the ImageNet dataset. As a result of the ILSVRC, major breakthroughs in computer vision have been accomplished. In 2012, AlexNet [97] was the first CNN to win the challenge with a top-5[1] error rate of 15.3 %. Figure D-1 shows the network architecture of AlexNet: the input image is fed into a combination of convolution and pooling layers and then through a series of densely connected layers where the final output is 1 of 1000 class labels. We can compare the network architecture to Figure 2-18, where the *feature learning* portion is performed by the cyan and lavender colored layers, and the *classification* portion is performed by the yellow colored layers.

AlexNet uses a single kernel of fixed size per convolutional layer. This yields good results if the relevant information (i.e. the object to be classified) within an image is always roughly in the same position and has the same size, which is not the case in real-life scenarios. A solution is to use multiple kernels of varying size per convolutional layer, effectively making the network *wider*. This idea has been implemented in the form of *inception modules* in GoogLeNet (also known as InceptionNet) [99] which won the ILSVRC in 2014 with a top-5 error rate of 6.67 %. Figure D-2 shows diagrams of the inception module, which places convolutions with kernels of varying size and a pooling operation in parallel. In the naïve version (Figure D-2a), the outputs of these operations are concatenated and serve as input to the next layer. and in the version with dimension reductions (Figure D-2b), additional 1×1 convolution operations are placed in series in order to decrease the depth of the feature maps and therefore the computational load per layer.

Another approach to improving classification performance is to simply build a deeper network. VGG Net [100], the second place in the 2014 ILSVRC, has achieved a top-5 error rate of 7.3 % by stacking more convolutional layers. Compared to the 5 convolutional layers used by AlexNet, VGG uses up to 19 3×3 convolutional layers.

Increasing network depth indefinitely comes with negative side-effects. Recall the backpropagation algorithm from Section 2-3-2: since neuron errors are defined as product of the derivative of the activation function, weight terms, and the neuron errors of the next layer, those neuron error terms can become very small in very deep networks. Since the neuron errors are directly related to the gradients computed to optimize the network's weight and bias terms,

[1] Top-5 refers to the fact that one of the five class labels deemed most likely by the network must match the actual label in order to be considered correct. In contrast, top-1 requires the most likely label to match the actual label. The top-5 error rate on ImageNet of a human expert is estimated to be about 5 % [76].

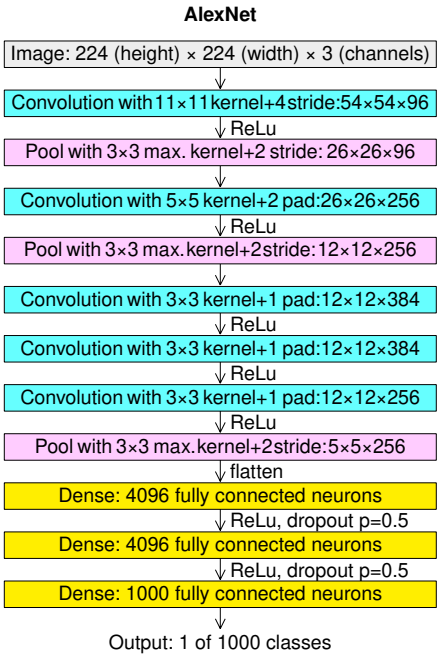
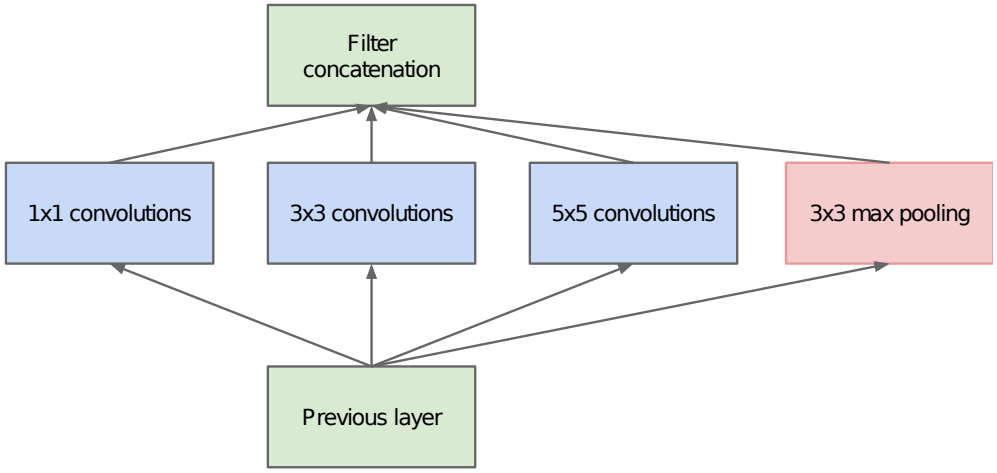


Figure D-1: Diagram of the AlexNet architecture. The network performs five convolution operations and three max-pooling operations with kernels of varying size. Note that these operations collectively correspond to the *feature learning* step as seen in Figure 2-18. Then, the feature representation is passed through three fully connected layers before the predicted class label is output. These operations correspond to the *classification* step in Figure 2-18. Image from [98].

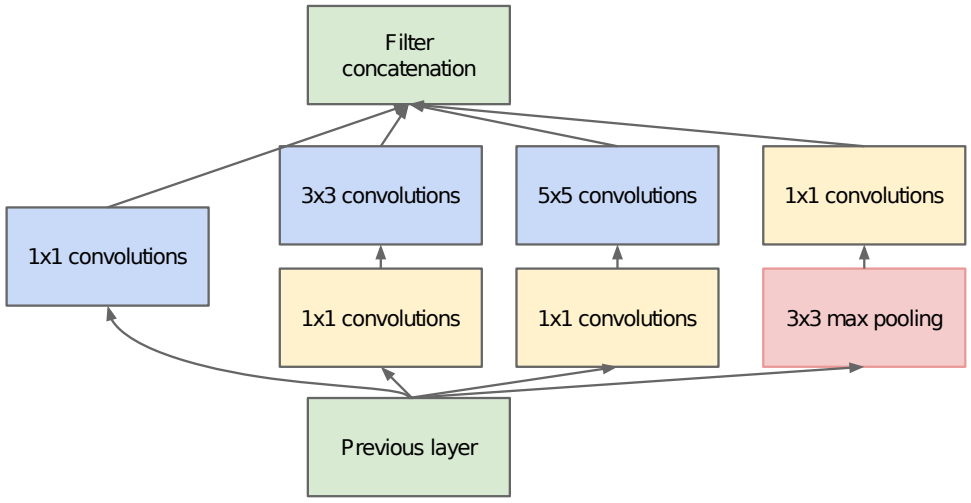
those gradients may become too small to be useful in the gradient descent algorithm: this is referred to as the **vanishing gradient problem**. Since the neuron errors are also defined by the derivative of the neuron's activation function, one solution to this problem is to use activation functions with piecewise constant derivatives such as Rectified Linear Unit (ReLU). Other solutions include improved weight initialization [101] as well as batch normalization [102] of inputs to individual layers.

Whereas the vanishing gradient problem occurs during the backpropagation stage of training, the feedforward stage suffers from a degradation problem as information has to pass through more layers in deeper networks, which causes deeper networks to perform worse than shallower counterparts beyond a certain amount of layers. Inspired by Long Short-Term Memory (LSTM) [103] and similar to Highway Networks [104], this degradation problem has been addressed by Residual Networks (ResNets) [48], which won the 2015 ILSVRC with a top-5 error rate of 3.57 %.

ResNet features *residual blocks* (see Figure D-3): The information $\mathbf{x}$ is allowed to skip over one or several layers, which are denoted to apply the function $\mathcal{F}(\mathbf{x})$. The outputs of the two paths are then added $\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathbf{x}$ and serve as input to the next residual block. Since the skipped connection is equivalent to an identity mapping, this leads to no added computational complexity and can be optimized using backpropagation and gradient descent. The network now learns feature mappings which represent a trade-off between transformations via convolution and direct passthrough of information. In the extreme case where an identity mapping between layers is ideal, $\mathcal{F}(\mathbf{x}) = \mathbf{0}$ and the effective depth of the network is reduced.



(a) Inception module, naïve version.



(b) Inception module with dimension reductions.

Figure D-2: Two implementations of the inception module. Rather than performing a single convolution or pooling operation per layer, an inception module performs convolution operations of varying size and a pooling operation in parallel, whose outputs get concatenated for the output of the module. Compared to the naïve version in **(a)**, additional 1×1 convolutions can be performed in series as shown in **(b)** in order to reduce the depth of the feature map and reduce computational complexity. Image from [99].

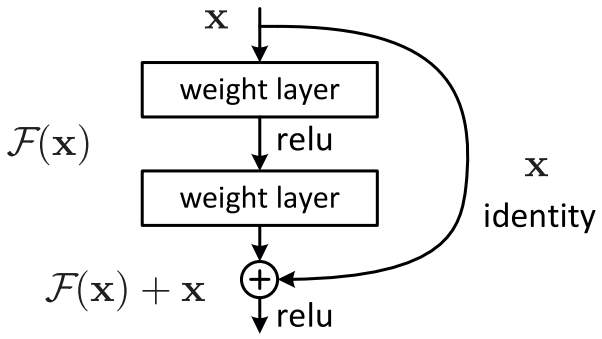


Figure D-3: A residual block as implemented in ResNet. The information $\mathbf{x}$ is allowed to skip over several layers where it is added to the output of the skipped layers $\mathcal{F}(\mathbf{x})$. As a result, the undisturbed input information $\mathbf{x}$ is allowed to travel much deeper into the network, which enables successful training of deeper networks compared to those without residual blocks. Image from [48].

The depth of ResNet networks is specified by adding the amount of convolutional layers and the amount of skip connections. For example, ResNet-50 has 34 convolutional layers and 16 skip connections.

By combining the ideas of inception modules from GoogLeNet and residual blocks from ResNet, the ResNeXt [105] architecture was developed, which won the 2016 ILSVRC with a top-5 error rate of 4.1 %. The concept of residual blocks has proven to be very useful for networks learning feature representations of images and modern state-of-the-art networks employ variations of this architecture, most notably EfficientNet [106] and RegNet [107]: EfficientNet is a scaling method that determines the width, depth, and resolution of networks like ResNet using a compound coefficient and RegNet performs neural architecture search (NAS) on various architectures including ResNet in order to find an optimal network for a given task.

D-3 Beyond CNNs: Transformer Networks and the Future of Computer Vision

In recent years, transformer-based networks have replaced CNNs as the top performers in ImageNet classification. Transformers, first introduced by Vaswani et al. in their 2017 paper "Attention Is All You Need" [79], have gained significant popularity due to their self-attention mechanism, which allows the model to focus on different parts of the input data in a more flexible manner. Originally designed for natural language processing (NLP) tasks, transformers have shown their potential in applications such as the powerful GPT-3 (Generative Pre-trained Transformer) language model [12]. However, transformers have been adapted for computer vision tasks as well, as demonstrated by the Vision Transformer (ViT) architecture [108].

The current top performer in ImageNet classification is Microsoft's Florence network [109], which integrates transformers in its architecture, achieving a top-5 error rate of 0.98 %. Other examples of networks integrating the ViT architecture are Meta AI's DINOv2 [110], which is a general-purpose network that generates robust feature representations of images across

a variety of domains, and Segment Anything (SAM) [80], which is a general-purpose image segmentation network.

Appendix E

Visual Example of Supervision Levels in Machine Learning

This appendix provides visual examples of supervised, unsupervised, semi-supervised, and self-supervised learning. All examples are based on the data distribution shown in Figure E-1a: a "half-circle" or "banana" distribution of data points with two class labels. This data could originate from a list of data points, such as the example of student grades from Section 2-2-2, or be the feature representation of image data that a convolutional neural network (CNN) generates (compare *feature learning* step in Figure 2-18.)

E-1 Supervised Learning

For supervised learning, the labels associated with every data point serve as the training signal: depending on whether the algorithm correctly or incorrectly determines a label during

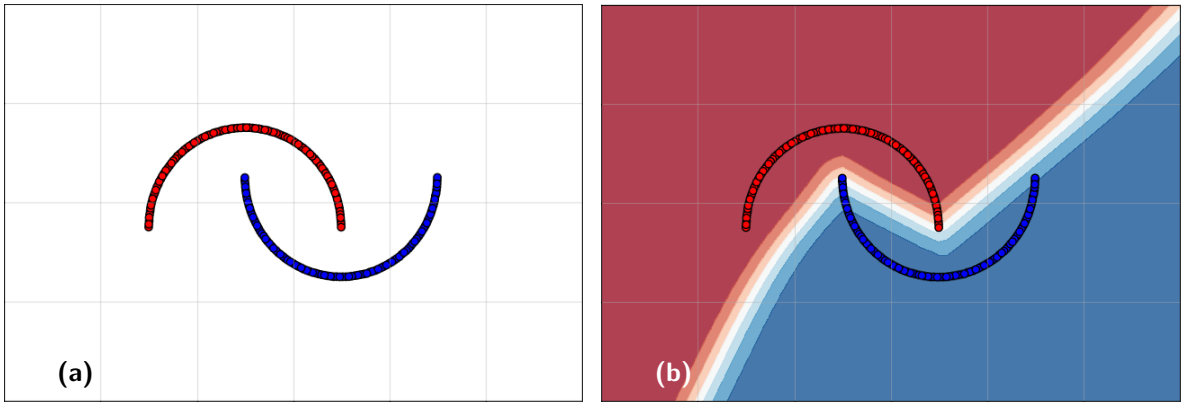


Figure E-1: Ideal distribution of a "half-circle" dataset **(a)** and a neural network classification boundary **(b)**.

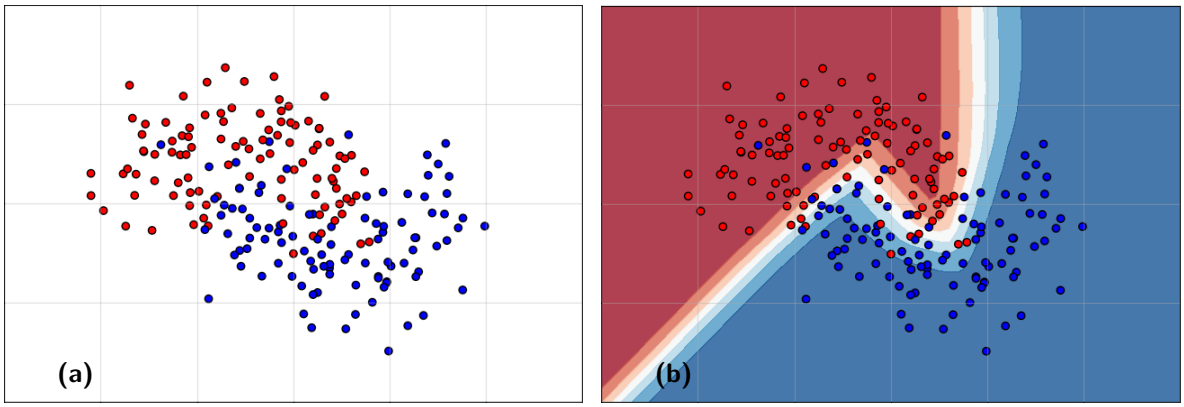


Figure E-2: (a): Noise corrupted distribution of the dataset from Figure E-1. Since there are many data points available, the classification boundary in **(b)** manages to separate the two classes relatively well.

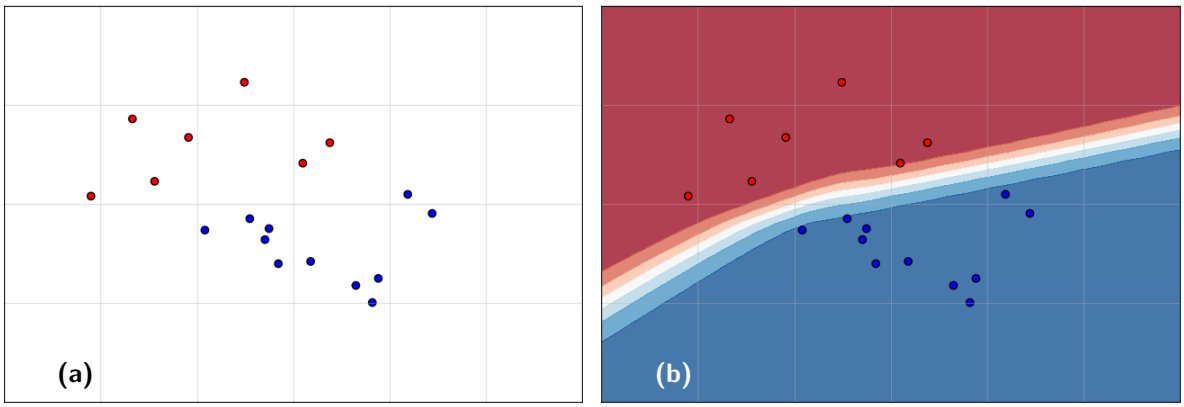


Figure E-3: (a): The same data distribution as shown in Figure E-2 but with only a few data points available. **(b):** The classifier trained on these few data points fails to generalize to the actual data distribution.

training, its internal parameters (weights) are adjusted accordingly such that its accuracy improves. If many labeled examples are available, this learning paradigm yields good results in a variety of tasks.

In Figure E-2a, a noise-corrupted distribution of the data from Figure E-1a is shown. Despite the noise, the learned classification boundary shown in Figure E-2b can separate the classes relatively well. In the case of a CNN, the network may not only learn to draw good classification boundaries, but also improve the representation of the data itself. In the extreme (and unrealistic) case, it may learn the ideal representation as shown in Figure E-1a, which would allow it to draw a very good classification boundary as shown in Figure E-2a.

Having more labeled data available will typically result in better performance. A counter-example is shown in Figure E-3a, where only a few data points are available. Training a classifier on these few data points (shown in Figure E-3b leads to underfitting: the classifier does not generalize well to the actual distribution which can be seen by the shape of the classification boundary strongly deviating from those in Figure E-2b or Figure E-1b.

The amount of labeled data that is required also depends on how representative the data is.

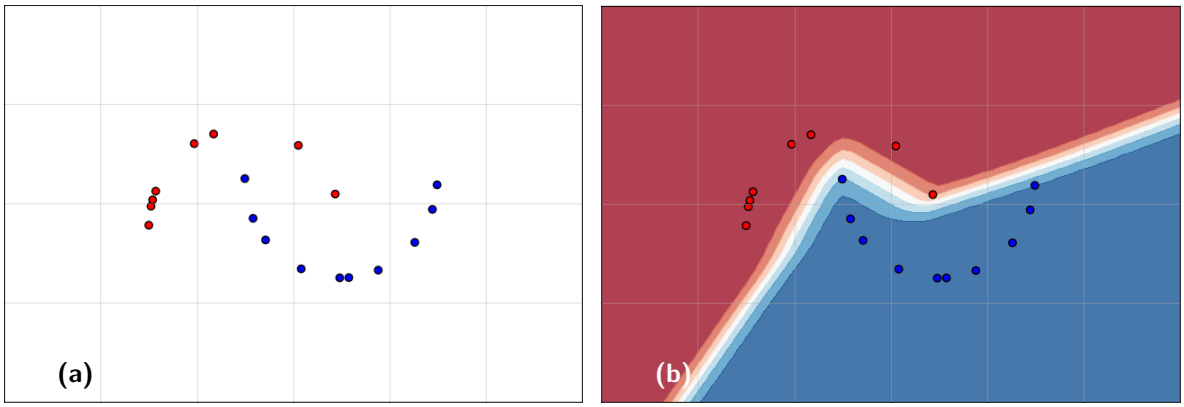


Figure E-4: (a): The ideal distribution from Figure E-1 with the same amount of data points as in Figure E-3. **(b):** Since the few data points that are available are very representative of the true distribution, the classifier performs better compared to Figure E-3b.

An example of this is shown in Figure E-4a where the same amount of data points is available as in Figure E-3a, but they are drawn from the true distribution. The corresponding classifier (Figure E-4b) performs much better since representation of the data is of higher quality. Of course, the data following the true distribution is not the only factor that improves the classification performance, but also that its sampling is representative of the true distribution: we can easily see that only drawing points from the rightmost part of the red half-circle and only from the leftmost part of the blue half-circle would lead to a very inaccurate classification boundary.

E-2 Unsupervised Learning

For unsupervised learning, no labels are available. The training signal is instead generated from the representation of the data. For example, in the case of a clustering algorithm, the training signal is determined by how easily the data points can be grouped into a pre-defined amount of clusters. An example clustering of the noise-corrupted data distribution into two clusters is shown in Figure E-5. Depending on error metrics of the particular clustering method used (e.g. the sum of distances of every point belonging to a cluster to the center of that cluster), a network can learn representations of the data that minimizes that error (e.g. it will learn to group data points in its feature representation).

The resulting distribution in Figure E-5d clearly shows two distinct clusters of data points. The obvious question that follows: Do these two clusters correspond to the two class labels that we had available in the supervised learning case? The answer is most likely no. Let us pretend that we had access to the labels all along and re-introduce them to the clustered distribution. This is shown in Figure E-6a: we can see that parts of the blue points are clustered with the majority of the red points an vice-versa. If we were to train a classifier on this distribution in a supervised manner, the performance may still be acceptable (Figure E-6b), but this is in no way guaranteed. In fact, if we do have labeled data available, naïve clustering of the data would most likely just unnecessarily decrease performance compared to simply training a classifier in a supervised fashion. The main takeaway is that unsupervised algorithms can perform their task fairly well, but the features that these algorithms focus

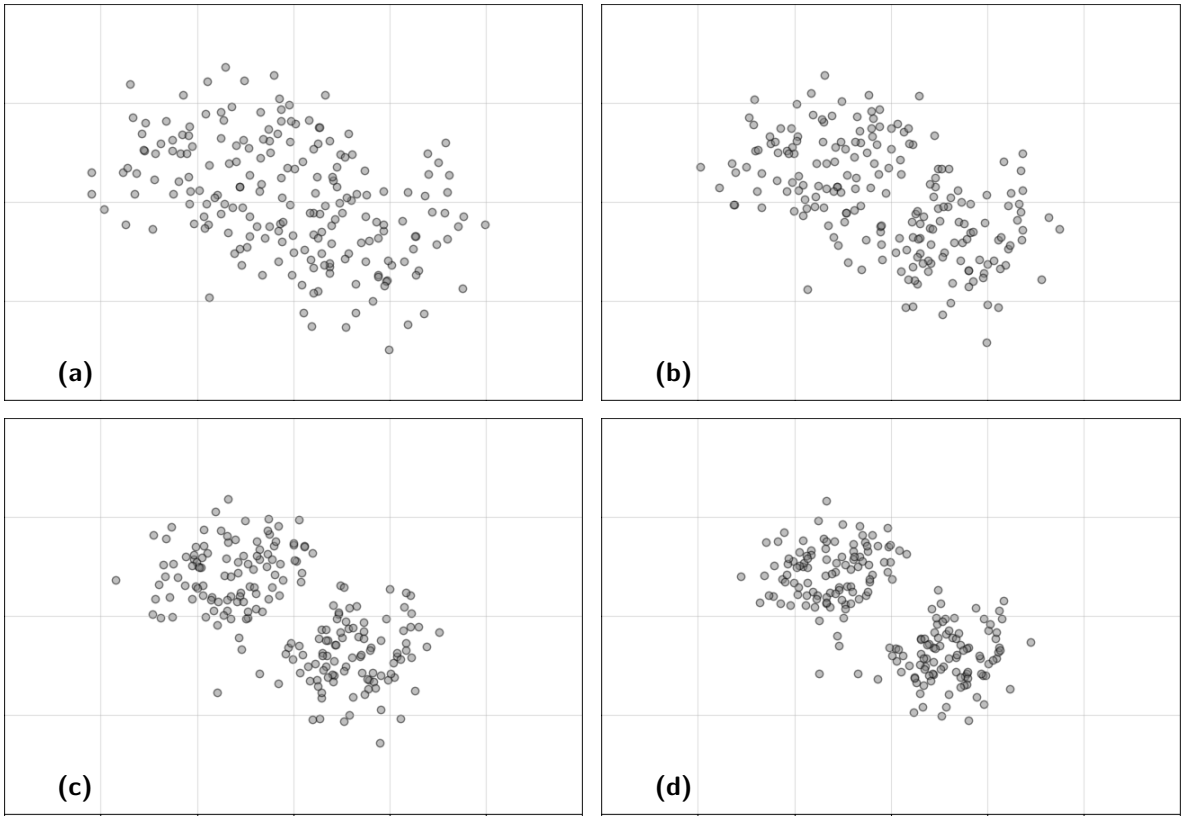


Figure E-5: (a): The same noise-corrupted distribution as in Figure E-2 but without any labels available. **(b)–(d):** Given the input that there should be 2 clusters, a clustering algorithm will learn a feature representation such that the data points are placed into 2 distinguishable clusters.

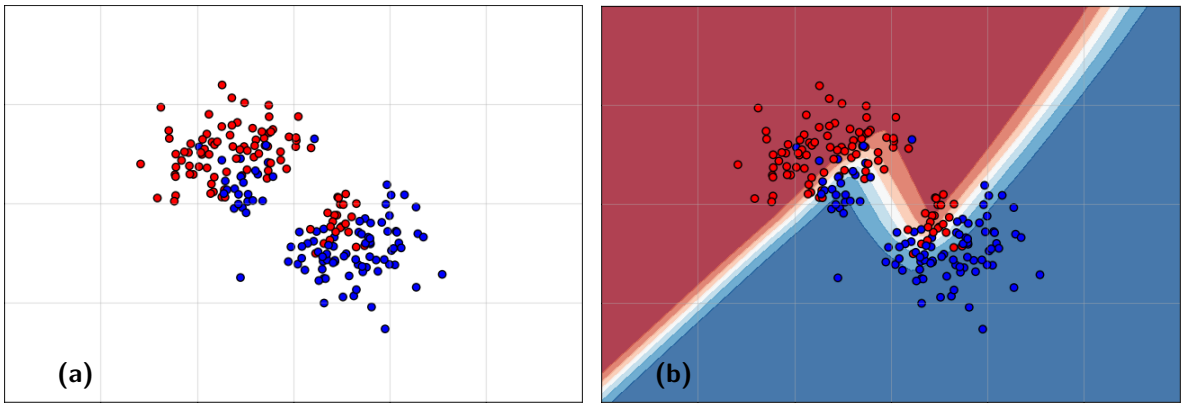


Figure E-6: (a): An example clustering similar to Figure E-5d but with the labels re-introduced. Although the clusters are well-defined, they do not correspond to the two class labels. **(b):** In this scenario, a classifier may still be able to separate the two classes relatively well, but in an unsupervised learning situation the necessary labels would not be available.

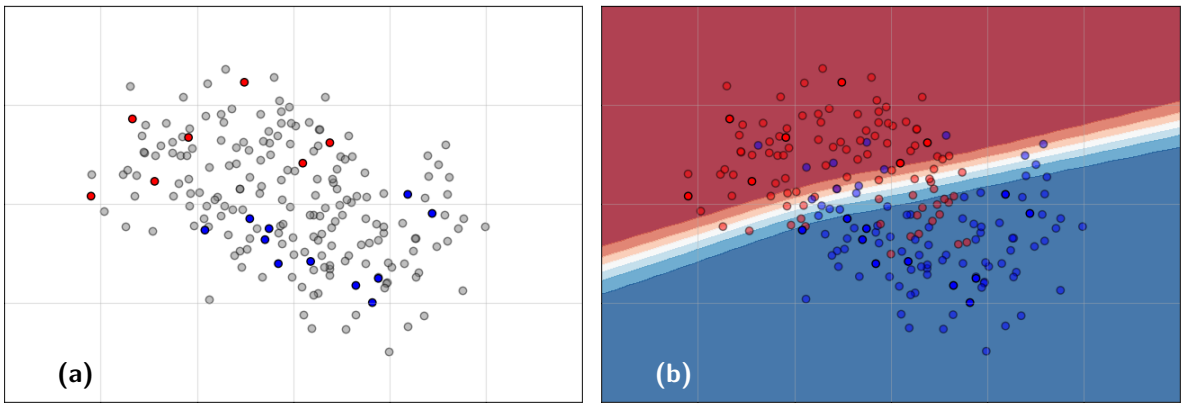


Figure E-7: (a): The same distribution as in Figure E-2 but only a few data points are labeled. The majority of data points are unlabeled and cannot be used for supervised learning. **(b):** A classifier trained on the labeled data points does not generalize to the actual data distribution. The true labels of the unlabeled data points are displayed slightly transparent.

on may not be the features that we are interested in. This is particularly important in the case of semi-supervised learning, where we attempt to combine unsupervised and supervised learning methods.

E-3 Semi-supervised Learning

For semi-supervised learning, only a few labeled data points may be available compared to a large amount of unlabeled ones. An example of this is shown in Figure E-7a. Similar to Figure E-3b, a classifier trained on these few labeled points will most likely overfit and not generalize well on the actual data (Figure E-7b).

However, what if we could shape the distribution of the data to match the true distribution as closely as possible? This ideally reshaped distribution is shown in Figure E-8a and its corresponding classification performance in Figure E-8b (compare also Figure E-4). If we

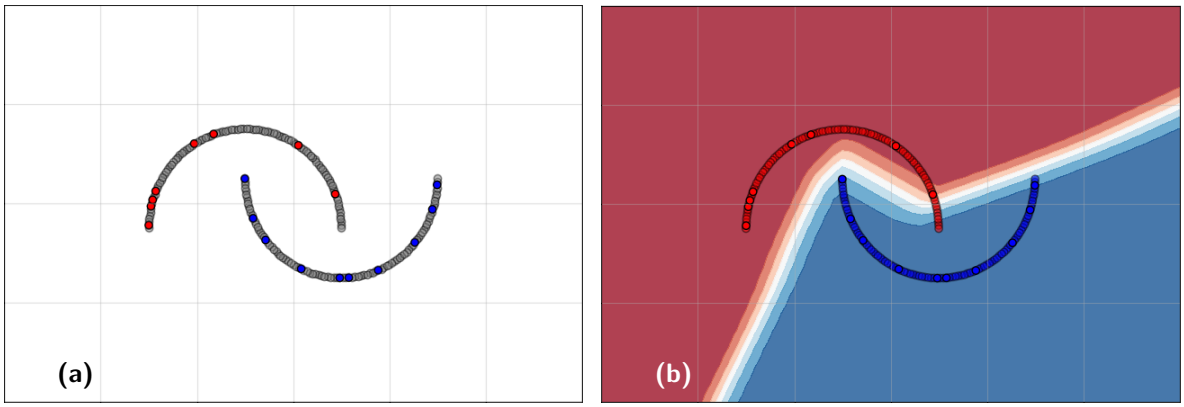


Figure E-8: (a): The same distribution as in Figure E-1 but only a few data points are labeled. The majority of data points are unlabeled and cannot be used for supervised learning. **(b):** Since the labeled data points are very representative of the true distribution, a classifier trained on these points performs much better than the one in Figure E-7b. The true labels of the unlabeled data points are displayed slightly transparent.

could train an algorithm to learn a high quality representation of the data, only a few labeled examples would suffice to train a classifier with acceptable performance. We have seen previously that an algorithm such as clustering may not be suitable for this, but there are algorithms that can achieve this goal.

E-4 Self-supervised Learning

Self-supervised learning algorithms are unsupervised learning algorithms. However, the training signal is not generated from characteristics of the learned representation of the data (e.g. clustering), but is instead dependent of the domain of the data and, in the case of a semi-supervised learning scheme, the downstream task we are interested in performing. This results in a pretext task (or pre-training task) that an unsupervised learning algorithm learns to perform on the input data. Examples of those pretext tasks are shown in Figure 2-19 (putting together a self-generated puzzle of an input image) and Figure 2-20 (learning that different views of the same input image should have similar representations). In terms of shaping the representation of the input data, the desired outcome is shown in Figure E-9: Rather than simply clustering the data, we would like to obtain a representation of the data that closely matches the true distribution (Figure E-9d). This is not straightforward, since the true distribution is usually not known. However, several self-supervised learning algorithms with pretext tasks tailored to a desired downstream task have been shown to be successful in improving feature representation of data and resulting in more efficient training when only a limited amount of labels are available [111, 13].

The increase in training efficiency with a limited amount of available labels is illustrated in Figure E-10: by introducing the labels after the self-supervised pre-training is completed, we can train a classifier with higher accuracy compared to having no pre-training performed. This is a combination of self-supervised and semi-supervised learning.

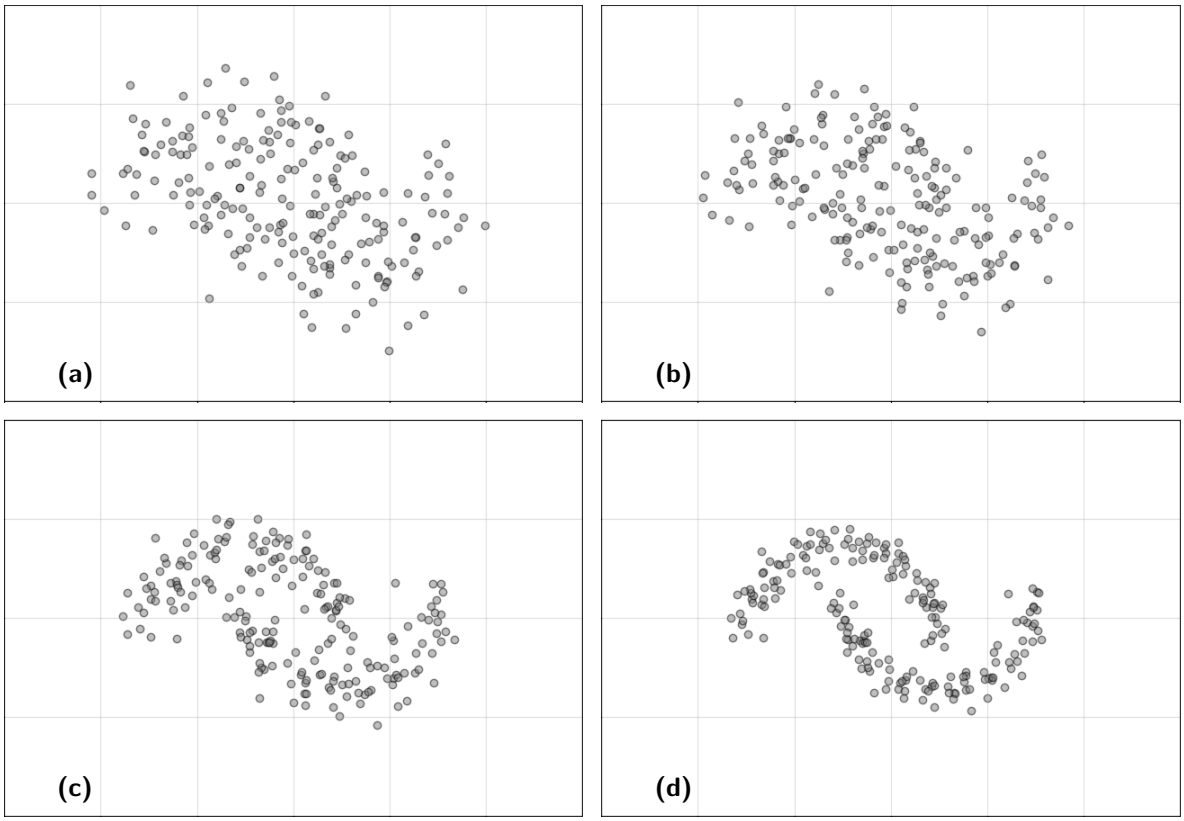


Figure E-9: (a): The same noise-corrupted distribution as in Figure E-2 but without any labels available. **(b)–(d):** Unlike a clustering algorithm that learns to group the data into a pre-defined number of clusters, a self-supervised learning algorithm may attempt to learn a representation of the data that is close to the true distribution (ideally).

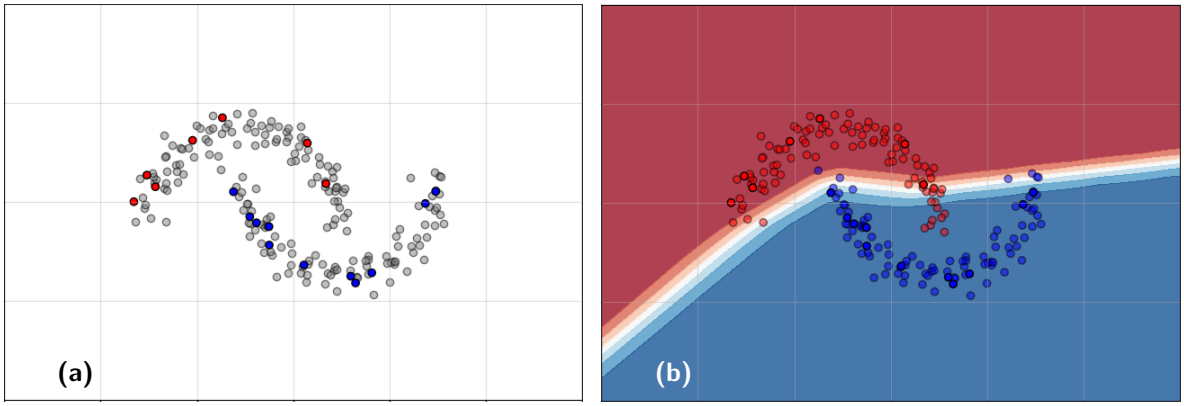
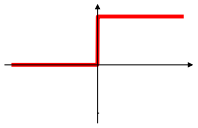
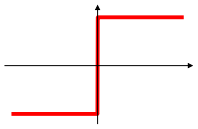
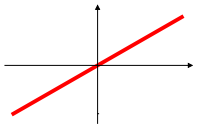
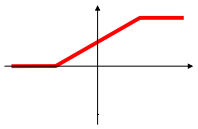
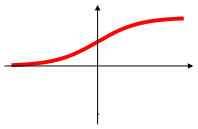
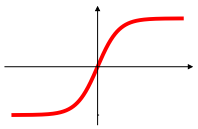
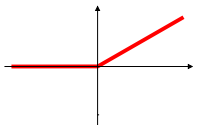
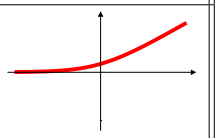


Figure E-10: (a): The resulting distribution of the example self-supervised learning algorithm from Figure E-9d with a few labeled data points available. **(b):** Since the distribution of the data is close to the true distribution (i.e. the learned representation is "good"), the few available labeled data points allow for a much more efficient training of a classifier. The true labels of the unlabeled data points are displayed slightly transparent.

Appendix F

Supplementary Tables and Figures

Table F-1: Typical activation functions used in neural networks. One main feature is that functions saturate, i.e., they approach a finite output value for very large input values. This is not always desired, in which case the ReLU function can be used. Many activation functions share similar properties but vary in smoothness and biasing. Note that the linear function is useful as activation function of an output layer for tasks like regression. Adapted from [112].

Activation function	Equation	Example application	1D graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0 \\ 0, & z = 0 \\ 1, & z > 0 \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 0, & z \leq -\frac{1}{2} \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2} \\ 1, & z \geq \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1+e^{-z}}$	Logistic regression, multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	Multi-layer NN	
Rectifier: ReLU	$\phi(z) = \max(0, z)$	Multi-layer NN	
Rectifier: softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer NN	

Bibliography

[1] World Health Organization, "Cancer," Mar. 2021. https://www.who.int/en/news-room/fact-sheets/detail/cancer Accessed: May 2023.

[2] National Cancer Institute, "Cancer staging," Mar. 2015. https://www.cancer.gov/about-cancer/diagnosis-staging/staging Accessed: May 2023.

[3] J. Galon, "Type, density, and location of immune cells within human colorectal tumors predict clinical outcome," *Science*, vol. 313, no. 5795, pp. 1960–1964, Sep. 2006. https://doi.org/10.1126/science.1129139

[4] D. Hammerl, J. W. M. Martens, M. Timmermans, M. Smid, A. M. Trapman-Jansen, R. Foekens, O. I. Isaeva, L. Voorwerk, H. E. Balcioglu, R. Wijers, I. Nederlof, R. Salgado, H. Horlings, M. Kok, and R. Debets, "Spatial immunophenotypes predict response to anti-PD1 treatment and capture distinct paths of T cell evasion in triple negative breast cancer," *Nature Communications*, vol. 12, no. 1, p. 5668, Sep. 2021. https://www.nature.com/articles/s41467-021-25962-0

[5] D. Bruni, H. K. Angell, and J. Galon, "The immune contexture and immunoscore in cancer prognosis and therapeutic efficacy," *Nature Reviews Cancer*, vol. 20, no. 11, pp. 662–680, Nov. 2020. https://doi.org/10.1038/s41568-020-0285-7

[6] A. Rahman, C. Jahangir, S. M. Lynch, N. Alattar, C. Aura, N. Russell, F. Lanigan, and W. M. Gallagher, "Advances in tissue-based imaging: impact on oncology research and clinical practice," *Expert Review of Molecular Diagnostics*, vol. 20, no. 10, pp. 1027–1037, 2020. https://doi.org/10.1080/14737159.2020.1770599

[7] H. E. Balcioglu, R. Wijers, D. Hammerl, M. Timmermans, M. Smid, A. M. Trapman-Jansen, J. Martens, and R. Debets, "TME Analyzer: a novel multi-dimensional visualization and analysis tool to identify immune contexture-based predictors in tumors," Dec. 2021, unpublished manuscript.

[8] C. C. Liu, C. B. Steen, and A. M. Newman, "Computational approaches for characterizing the tumor immune microenvironment," *Immunology*, vol. 158, no. 2, pp. 70–84, 2019. https://onlinelibrary.wiley.com/doi/abs/10.1111/imm.13101

[9] E. Meijering, "A bird's-eye view of deep learning in bioimage analysis," *Computational and Structural Biotechnology Journal*, vol. 18, pp. 2312–2325, 2020. https://www.sciencedirect.com/science/article/pii/S2001037020303561

[10] E. Moen, D. Bannon, T. Kudo, W. Graf, M. Covert, and D. Van Valen, "Deep learning for cellular image analysis," *Nature Methods*, vol. 16, no. 12, pp. 1233–1246, Dec. 2019. https://doi.org/10.1038/s41592-019-0403-1

[11] Z. Liu, L. Jin, J. Chen, Q. Fang, S. Ablameyko, Z. Yin, and Y. Xu, "A survey on applications of deep learning in microscopy image analysis," *Computers in Biology and Medicine*, vol. 134, p. 104523, Jul. 2021. https://www.sciencedirect.com/science/article/pii/S0010482521003176

[12] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *arXiv*, Jul. 2020. https://arxiv.org/abs/2005.14165

[13] R. Balestriero, M. Ibrahim, V. Sobal, A. Morcos, S. Shekhar, T. Goldstein, F. Bordes, A. Bardes, G. Mialon, Y. Tian, A. Schwarzschild, A. G. Wilson, J. Geiping, Q. Garrido, P. Fernandez, A. Bar, H. Pirsiavash, Y. LeCun, and M. Goldblum, "A cookbook of self-supervised learning," *arXiv*, Apr. 2023. https://arxiv.org/abs/2304.12210

[14] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A Simple Framework for Contrastive Learning of Visual Representations," *arXiv*, Jun. 2020. http://arxiv.org/abs/2002.05709

[15] R. Krishnan, P. Rajpurkar, and E. J. Topol, "Self-supervised learning in medicine and healthcare," *Nature Biomedical Engineering*, vol. 6, no. 12, pp. 1346–1352, Dec 2022. https://doi.org/10.1038/s41551-022-00914-1

[16] O. Ciga, T. Xu, and A. L. Martel, "Self supervised contrastive learning for digital histopathology," *arXiv*, Sep. 2021. http://arxiv.org/abs/2011.13971

[17] N. S. Punn and S. Agarwal, "BT-Unet: A self-supervised learning framework for biomedical image segmentation using Barlow Twins with U-Net models," *arXiv*, Mar. 2022. http://arxiv.org/abs/2112.03916

[18] D. Jiménez-Sánchez, M. Ariz, H. Chang, X. Matias-Guiu, C. E. de Andrea, and C. O. de Solórzano, "Naronet: Discovery of tumor microenvironment elements from highly multiplexed images," *Medical Image Analysis*, vol. 78, p. 102384, 2022. https://www.sciencedirect.com/science/article/pii/S1361841522000366

[19] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, "What Makes for Good Views for Contrastive Learning?" *arXiv*, Dec. 2020. http://arxiv.org/abs/2005.10243

[20] T. D. Hewitson, B. Wigg, and G. J. Becker, "Tissue preparation for histochemistry: Fixation, embedding, and antigen retrieval for light microscopy," in *Histology Protocols*, T. D. Hewitson and I. A. Darby, Eds. Totowa, NJ: Humana Press, 2010, pp. 3–18. https://doi.org/10.1007/978-1-60327-345-9_1

[21] J. D. Bancroft and C. Layton, "The hematoxylins and eosin," in *Bancroft's Theory and Practice of Histological Techniques*, 7th ed., S. K. Suvarna, C. Layton, and J. D. Bancroft, Eds. Oxford: Churchill Livingstone, Jan. 2013, pp. 173–186. https://www.sciencedirect.com/science/article/pii/B978070204226300010X

[22] J. A. Ramos-Vara and M. A. Miller, "When Tissue Antigens and Antibodies Get Along: Revisiting the Technical Aspects of Immunohistochemistry—The Red, Brown, and Blue Technique," *Veterinary Pathology*, vol. 51, no. 1, pp. 42–87, Jan. 2014. https://doi.org/10.1177/0300985813505879

[23] E. R. Parra, N. Uraoka, M. Jiang, P. Cook, D. Gibbons, M.-A. Forget, C. Bernatchez, C. Haymaker, I. I. Wistuba, and J. Rodriguez-Canales, "Validation of multiplex immunofluorescence panels using multispectral microscopy for immune-profiling of formalin-fixed and paraffin-embedded human tumor tissues," *Scientific Reports*, vol. 7, p. 13380, Oct. 2017. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5645415/

[24] E. J. Groen, J. Hudecek, L. Mulder, M. van Seijen, M. M. Almekinders, S. Alexov, A. Kovács, A. Ryska, Z. Varga, F.-J. Andreu Navarro, S. Bianchi, W. Vreuls, E. Balslev, M. V. Boot, J. Kulka, E. Chmielik, E. Barbé, M. J. de Rooij, W. Vos, A. Farkas, N. E. Leeuwis-Fedorovich, P. Regitnig, P. J. Westenend, L. F. S. Kooreman, C. Quinn, G. Floris, G. Cserni, P. J. van Diest, E. H. Lips, M. Schaapveld, J. Wesseling, and Grand Challenge PRECISION consortium, "Prognostic value of histopathological DCIS features in a large-scale international interrater reliability study," *Breast Cancer Research and Treatment*, vol. 183, no. 3, pp. 759–770, Oct. 2020. https://doi.org/10.1007/s10549-020-05816-x

[25] "Opal 7-Color Kit For Multiplex IHC | Akoya Biosciences." https://www.akoyabio.com/phenoimager/assays/opal-7-color-ihc-kit/ Accessed: May 2023.

[26] J. Galon and D. Bruni, "Tumor immunology and tumor evolution: Intertwined histories," *Immunity*, vol. 52, no. 1, pp. 55–81, Jan. 2020. https://www.sciencedirect.com/science/article/pii/S1074761319305370

[27] N. M. Anderson and M. C. Simon, "The tumor microenvironment," *Current Biology*, vol. 30, no. 16, pp. R921–R925, Aug. 2020. https://doi.org/10.1016/j.cub.2020.06.081

[28] R. Moll, W. W. Franke, D. L. Schiller, B. Geiger, and R. Krepler, "The catalog of human cytokeratins: Patterns of expression in normal epithelia, tumors and cultured cells," *Cell*, vol. 31, no. 1, pp. 11–24, Nov. 1982. https://doi.org/10.1016/0092-8674(82)90400-7

[29] D. Vasilevska, V. Rudaitis, A. Adamiak-Godlewska, A. Semczuk-Sikora, D. Lewkowicz, D. Vasilevska, and A. Semczuk, "Cytokeratin expression pattern in human endometrial carcinomas and lymph nodes micrometastasis: a mini-review," *Journal of Cancer*, vol. 13, no. 6, pp. 1713–1724, 2022. https://doi.org/10.7150/jca.70550

[30] H. Zola, B. Swart, A. Banham, S. Barry, A. Beare, A. Bensussan, L. Boumsell, C. D. Buckley, H.-J. Bühring, G. Clark, P. Engel, D. Fox, B.-Q. Jin, P. J. Macardle, F. Malavasi, D. Mason, H. Stockinger, and X. Yang, "Cd molecules 2006 — human cell differentiation molecules," *Journal of Immunological Methods*, vol. 319, no. 1, pp. 1–5, Jan. 2007. https://www.sciencedirect.com/science/article/pii/S002217590600336X

[31] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[33] B. Timmermans, "Machine learning overview," 2017. https://web.archive.org/web/20190423035419/http://btimmermans.com/wp-content/uploads/2017/12/machine_learning_overview.jpg Accessed: Jul. 2022.

[34] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," *PLOS ONE*, vol. 10, no. 3, p. e0118432, Mar. 2015. https://doi.org/10.1371/journal.pone.0118432

[35] M. Verhaegen and V. Verdult, *Filtering and System Identification: A Least Squares Approach*. Cambridge University Press, 2007. https://doi.org/10.1017/CBO9780511618888

[36] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2nd ed. Springer, 2009. http://www-stat.stanford.edu/~tibs/ElemStatLearn/

[37] A. M. Rufai, "Generative vs. Discriminative Models in Machine Learning," Aug. 2020. https://betterprogramming.pub/generative-vs-discriminative-models-d26def8fd64a Accessed: May 2023.

[38] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. https://www.sciencedirect.com/science/article/pii/0893608089900208

[39] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, Jan. 1993. https://www.sciencedirect.com/science/article/pii/S0893608005801315

[40] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017. https://arxiv.org/abs/1710.05941

[41] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. http://neuralnetworksanddeeplearning.com

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. https://arxiv.org/abs/1412.6980

[43] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011. https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[44] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[45] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," 2017. https://arxiv.org/abs/1708.03888

[46] A. Amini and A. Soleimany, "MIT Deep Learning 6.S191." http://introtodeeplearning.com Accessed: May 2023.

[47] D. Cornelisse, "An intuitive guide to Convolutional Neural Networks," Apr. 2018. https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/ Accessed: Jul. 2022.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv*, Dec. 2015. http://arxiv.org/abs/1512.03385

[49] Y. LeCun and I. Misra, "Self-supervised learning: The dark matter of intelligence." https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/ Accessed: Apr. 2023.

[50] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. https://doi.org/10.1038/nature14539

[51] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum Contrast for Unsupervised Visual Representation Learning," *arXiv*, Mar. 2020. http://arxiv.org/abs/1911.05722

[52] X. Chen, H. Fan, R. Girshick, and K. He, "Improved Baselines with Momentum Contrastive Learning," *arXiv*, Mar. 2020. http://arxiv.org/abs/2003.04297

[53] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton, "Big Self-Supervised Models are Strong Semi-Supervised Learners," *arXiv*, Oct. 2020. http://arxiv.org/abs/2006.10029

[54] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised Visual Representation Learning by Context Prediction," *arXiv*, Jan. 2016. http://arxiv.org/abs/1505.05192

[55] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, "Bootstrap your own latent: A new approach to self-supervised Learning," *arXiv*, Sep. 2020. http://arxiv.org/abs/2006.07733

[56] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised Learning of Visual Features by Contrasting Cluster Assignments," *arXiv*, Jan. 2021. http://arxiv.org/abs/2006.09882

[57] P. Goyal, M. Caron, B. Lefaudeux, M. Xu, P. Wang, V. Pai, M. Singh, V. Liptchinsky, I. Misra, A. Joulin, and P. Bojanowski, "Self-supervised Pretraining of Visual Features in the Wild," *arXiv*, Mar. 2021. http://arxiv.org/abs/2103.01988

[58] T. Chen, "Advancing Self-Supervised and Semi-Supervised Learning with SimCLR." http://ai.googleblog.com/2020/04/advancing-self-supervised-and-semi.html Accessed: Jul. 2022.

[59] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv*, Jan. 2019. https://arxiv.org/abs/1807.03748

[60] W. Ågren, "The NT-Xent loss upper bound," 2022. https://arxiv.org/abs/2205.03169

[61] Y. LeCun, C. Cortes, and C. J. C. Burges, "THE MNIST DATABASE of handwritten digits," 1998. http://yann.lecun.com/exdb/mnist/ Accessed: May 2023.

[62] L. Jing, P. Vincent, Y. LeCun, and Y. Tian, "Understanding Dimensional Collapse in Contrastive Self-supervised Learning," *arXiv*, Apr. 2022. http://arxiv.org/abs/2110.09348

[63] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 649–666. https://doi.org/10.1007/978-3-319-46487-9_40

[64] R. Zhang, P. Isola, and A. A. Efros, "Split-brain autoencoders: Unsupervised learning by cross-channel prediction," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jul. 2017. https://doi.org/10.1109/cvpr.2017.76

[65] P. Bankhead, M. B. Loughrey, J. A. Fernández, Y. Dombrowski, D. G. McArt, P. D. Dunne, S. McQuaid, R. T. Gray, L. J. Murray, H. G. Coleman, J. A. James, M. Salto-Tellez, and P. W. Hamilton, "QuPath: Open source software for digital pathology image analysis," *Scientific Reports*, vol. 7, no. 1, p. 16878, Dec. 2017. https://www.nature.com/articles/s41598-017-17204-5

[66] D. R. Stirling, A. E. Carpenter, and B. A. Cimini, "CellProfiler Analyst 3.0: accessible data exploration and machine learning for image analysis," *Bioinformatics*, vol. 37, no. 21, pp. 3992–3994, 09 2021. https://doi.org/10.1093/bioinformatics/btab634

[67] "ImageJ Wiki." https://imagej.github.io/ Accessed: May 2023.

[68] E. T. Dobson, B. Cimini, A. H. Klemm, C. Wählby, A. E. Carpenter, and K. W. Eliceiri, "ImageJ and CellProfiler: Complements in open-source bioimage analysis," *Current Protocols*, vol. 1, no. 5, May 2021. https://doi.org/10.1002/cpz1.89

[69] F. Meyer, "The watershed concept and its use in segmentation : a brief history," *arXiv*, Feb. 2012. http://arxiv.org/abs/1202.0216

[70] U. Schmidt, M. Weigert, C. Broaddus, and G. Myers, "Cell Detection with Star-convex Polygons," *arXiv*, vol. 11071, pp. 265–273, 2018. http://arxiv.org/abs/1806.03535

[71] "inForm Automated Image Analysis Software." https://www.akoyabio.com/phenoimager/software/inform-tissue-finder/ Accessed: May 2023.

[72] *inForm® User Manual*. https://resources.perkinelmer.com/corporate/content/lst_software_downloads/informusermanual_2_3_0_rev1.pdf

[73] C. Stringer, T. Wang, M. Michaelos, and M. Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," *Nature Methods*, vol. 18, no. 1, pp. 100–106, Jan. 2021. https://www.nature.com/articles/s41592-020-01018-x

[74] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 215–223. https://proceedings.mlr.press/v15/coates11a.html

[75] "ImageNet." https://image-net.org/ Accessed: May 2023.

[76] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *arXiv*, Jan. 2015. http://arxiv.org/abs/1409.0575

[77] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" *arXiv*, Jun. 2019. https://arxiv.org/abs/1805.08974

[78] L. Keren, M. Bosse, S. Thompson, T. Risom, K. Vijayaragavan, E. McCaffrey, D. Marquez, R. Angoshtari, N. F. Greenwald, H. Fienberg, J. Wang, N. Kambham, D. Kirkwood, G. Nolan, T. J. Montine, S. J. Galli, R. West, S. C. Bendall, and M. Angelo, "MIBI-TOF: A multiplexed imaging platform relates cellular phenotypes and tissue structure," *Science Advances*, vol. 5, no. 10, Oct. 2019. https://advances.sciencemag.org/content/5/10/eaax5851

[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. https://arxiv.org/abs/1706.03762

[80] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," *arXiv*, Apr. 2023. https://doi.org/10.48550/arXiv.2304.02643

[81] T. Toth, D. Bauer, F. Sukosd, and P. Horvath, "Fisheye transformation enhances deep-learning-based single-cell phenotyping by including cellular microenvironment," *Cell Reports Methods*, vol. 2, no. 12, p. 100339, Dec. 2022. https://doi.org/10.1016/j.crmeth.2022.100339

[82] J. Li, P. Zhou, C. Xiong, and S. C. H. Hoi, "Prototypical contrastive learning of unsupervised representations," 2020. https://arxiv.org/abs/2005.04966

[83] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow Twins: Self-Supervised Learning via Redundancy Reduction," *arXiv*, Jun. 2021. http://arxiv.org/abs/2103.03230

[84] A. Bardes, J. Ponce, and Y. LeCun, "Vicreg: Variance-invariance-covariance regularization for self-supervised learning," 2021. https://arxiv.org/abs/2105.04906

[85] J.-R. Lin, M. Fallahi-Sichani, J.-Y. Chen, and P. K. Sorger, "Cyclic immunofluorescence (CycIF), a highly multiplexed method for single-cell imaging," *Current Protocols in Chemical Biology*, vol. 8, no. 4, pp. 251–264, Dec. 2016. https://doi.org/10.1002/cpch.14

[86] V. L'Imperio, E. Wulczyn, M. Plass, H. Müller, N. Tamini, L. Gianotti, N. Zucchini, R. Reihs, G. S. Corrado, D. R. Webster, L. H. Peng, P.-H. C. Chen, M. Lavitrano, Y. Liu, D. F. Steiner, K. Zatloukal, and F. Pagni, "Pathologist Validation

of a Machine Learning–Derived Feature for Colon Cancer Risk Stratification," *JAMA Network Open*, vol. 6, no. 3, pp. e2 254 891–e2 254 891, Mar. 2023. https://doi.org/10.1001/jamanetworkopen.2022.54891

[87] "Vectra 3 Automated Quantitative Pathology Imaging System." https://www.akoyabio.com/phenoimager/instruments/vectra-3-0/ Accessed: May 2023.

[88] Delft University of Technology, "Combination of microscopy techniques makes images twice as sharp." https://www.tudelft.nl/en/2019/tnw/combination-of-microscopy-techniques-makes-images-twice-as-sharp Accessed: Jul. 2022.

[89] R. Heintzmann and U. Kubitscheck, "Introduction to Optics," in *Fluorescence Microscopy*. John Wiley & Sons, Ltd, 2017, ch. 1, pp. 1–22. https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527687732.ch1

[90] DrBob, "A positive lens." Mar. 2006. https://commons.wikimedia.org/wiki/File:Lens1.svg CC-BY-SA 3.0 Accessed: May 2023.

[91] U. Kubitscheck, "Principles of Light Microscopy," in *Fluorescence Microscopy*. John Wiley & Sons, Ltd, 2017, ch. 2, pp. 23–83. https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527687732.ch2

[92] S. Bliven, "Two airy disks at various spacings," Mar. 2014. https://commons.wikimedia.org/wiki/File:Airy_disk_spacing_near_Rayleigh_criterion.png Accessed: Jul. 2022.

[93] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998. https://doi.org/10.1109/5.726791

[94] S. An, M. Lee, S. Park, H. Yang, and J. So, "An Ensemble of Simple Convolutional Neural Network Models for MNIST Digit Recognition," *arXiv*, Oct. 2020. http://arxiv.org/abs/2008.10400

[95] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," *arXiv*, Feb. 2015. http://arxiv.org/abs/1405.0312

[96] "COCO - Common Objects in Context." https://cocodataset.org/ Accessed: Jul. 2022.

[97] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012. https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[98] CMG Lee, "Comparison of the LeNet and AlexNet convolution, pooling, and dense layers." https://commons.wikimedia.org/wiki/File:Comparison_image_neural_networks.svg CC-BY-SA 4.0 Accessed: May 2023.

[99] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv*, Sep. 2014. http://arxiv.org/abs/1409.4842

[100] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv*, Apr. 2015. http://arxiv.org/abs/1409.1556

[101] Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient BackProp," in *Neural Networks: Tricks of the trade*, G. Orr and M. K., Eds. Springer, 1998. https://doi.org/10.1007/978-3-642-35289-8_3

[102] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv*, Mar. 2015. http://arxiv.org/abs/1502.03167

[103] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[104] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway Networks," *arXiv*, Nov. 2015. http://arxiv.org/abs/1505.00387

[105] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," *arXiv*, Apr. 2017. http://arxiv.org/abs/1611.05431

[106] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *arXiv*, Sep. 2020. http://arxiv.org/abs/1905.11946

[107] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing Network Design Spaces," *arXiv*, Mar. 2020. http://arxiv.org/abs/2003.13678

[108] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv*, Jun. 2021. http://arxiv.org/abs/2010.11929

[109] L. Yuan, D. Chen, Y.-L. Chen, N. Codella, X. Dai, J. Gao, H. Hu, X. Huang, B. Li, C. Li, C. Liu, M. Liu, Z. Liu, Y. Lu, Y. Shi, L. Wang, J. Wang, B. Xiao, Z. Xiao, J. Yang, M. Zeng, L. Zhou, and P. Zhang, "Florence: A New Foundation Model for Computer Vision," *arXiv*, Nov. 2021. http://arxiv.org/abs/2111.11432

[110] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "Dinov2: Learning robust visual features without supervision," *arXiv*, 2023. https://arxiv.org/abs/2304.07193

[111] G. Huang, I. Laradji, D. Vazquez, S. Lacoste-Julien, and P. Rodriguez, "A Survey of Self-Supervised and Few-Shot Object Detection," *arXiv*, Nov. 2021. http://arxiv.org/abs/2110.14711

[112] S. Raschka, "Activation functions for artificial neural networks," 2020. http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/#activation-functions-for-artificial-neural-networks Accessed: Jul. 2022.

Glossary

List of Acronyms

TU Delft Delft University of Technology

Erasmus MC Erasmus University Medical Center

ACE TI-IT Academic Center of Excellence for Tumor Immunology and Immune Therapy

AI Artificial Intelligence

ANN artificial neural network

CD Clusters of Differentiation

CIFAR Canadian Institute For Advanced Research

CK cytokeratin

CNN convolutional neural network

COCO Common Objects in Context

DAPI 4',6-diamidino-2-phenylindole

DL Deep Learning

FN false negative

FP false positive

GPU graphics processing unit

H&E hematoxylin and eosin

IHC immunohistochemistry

i.i.d. independent and identically distributed

ILSVRC ImageNet Large Scale Visual Recognition Challenge

InfoNCE Information Noise-Contrastive Estimation

LARS Layer-wise Adaptive Rate Scaling

LSTM Long Short-Term Memory

MAP maximum a posteriori probability

MIBI-TOF multiplexed ion beam imaging by time of flight
MLP multilayer perceptron
ML Machine Learning
MNIST Modified National Institute of Standards and Technology
MoM mean of medians
MxIF multiplex immunofluorescence
NAS neural architecture search
NN neural network
NT-Xent normalized temperature cross-entropy
PCL patch-contrastive learning
PR-AUC Area Under the Precision-Recall Curve
PR Precision-Recall
ReLU Rectified Linear Unit
ResNet Residual Network
RGB red green blue
ROC-AUC Area Under the Receiver Operating Characteristic Curve
ROC Receiver Operating Characteristic
SGD Stochastic Gradient Descent
SimCLR Simple Framework for Contrastive Learning of Visual Representations
SLP single-layer perceptron
TME tumor microenvironment
TNBC triple-negative breast cancer
TNM tumor-node-metastasis
TN true negative
TP true positive

List of Symbols

$\boldsymbol{\delta}^{(l)}$ Vector of neuron errors in layer l of a neural network
$\boldsymbol{\theta}$ Optimization variable vector (see also $\mathbf{w}$)
$\delta_j^{(l)}$ Error of the jth neuron in layer l of a neural network
η Learning rate
λ Regularization weight (machine learning)
λ Wavelength of light
∇_a Gradient vector operator for derivatives with respect to a
$\phi(\cdot)$ Activation function

τ	Temperature parameter (NT-Xent loss function)
θ	Half angular aperture (optics)
θ_j	Optimization variable (see also w_i)
$\boldsymbol{h}$	Feature representation (SimCLR)
$\boldsymbol{x}$	Input image (SimCLR)
$\boldsymbol{z}$	Feature projection (SimCLR)
$\|\cdot\|_p$	Mathematical norm of order p
$\mathbf{1}$	Ones vector
$\mathbf{1}_{[k\neq i]}$	Indicator function that is equal to 1 iff $k \neq i$ and 0 otherwise
$\mathbf{A} \circ \mathbf{B}$	Hadamard product of $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{e}(\cdot)$	Error vector function
$\mathbf{I}$	Identity matrix
$\mathbf{w}$	Weight vector
$\mathbf{W}^{(l)}$	Matrix of weights of connections between neurons of layers l and $l-1$
$\mathbf{X}$	Data matrix
$\mathbf{x}$	Data vector
$\mathbf{y}$	Observation vector
$\mathbf{z}^{(l)}$	Vector of weighted inputs to the activation functions of neurons in layer l
$\mathcal{L}^{\text{NT-Xent}}$	NT-Xent loss function
$\mathcal{L}_{\text{A}}$	Alignment loss (NT-Xent loss function)
$\mathcal{L}_{\text{D}}$	Distribution loss (NT-Xent loss function)
$\mathcal{MB}$	Minibatch (set)
$\mathcal{T}$	Set of transformations (SimCLR)
NA	Numerical aperture
$a_j^{(l)}$	Activation of the jth neuron in layer l of a neural network
B	Image (geometrical optics)
$b_j^{(l)}$	Bias of the jth neuron in layer l of a neural network
C	Contrast
d	Lens thickness
d_{R}	Rayleigh resolution distance
f	Focal length
I	Light intensity (optics)
$J(\cdot)$	Cost function or loss function
$J_1(\cdot)$	Bessel function of the first kind
M	Magnification
m	Degree of a model (e.g. polynomial)
N	Batch size, sample size
n	Refractive index
O	Object (geometrical optics)
$p(x)$	Probability of x

$p(x \mid y)$	Probability of x given y
R	Lens radius
t	Transformation (SimCLR)
w_i	Weight coefficient
$w_{jk}^{(l)}$	Weight of connection between the j th neuron in layer l and the kth neuron in layer $l-1$ of a neural network
$z_j^{(l)}$	Sum of weighted inputs to the activation function of the jth neuron in layer l of a neural network