Tools for Conversion, Analysis, and Annotation of Histology Images

by

Johnathan Pocock

Thesis

Under supervision of

Professor Nasir Rajpoot and Dr Fayyaz Minhas

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Department of Computer Science

July 2023
Contents

Abstract ................................................................. viii
Acknowledgements .................................................. x
Abbreviations ......................................................... xii
List of Figures ......................................................... xviii
List of Tables ......................................................... xx
List of Listings ......................................................... xxi

1 Introduction .......................................................... 1

1.1 The Histopathology Pipeline ................................. 3
  1.1.1 Clinical Terms ............................................. 4
  1.1.2 High-Level Overview of the Pathology Pipeline .... 5
  1.1.3 Fixing & Embedding .................................... 5
  1.1.4 Sectioning ................................................ 9
  1.1.5 Staining ................................................... 10
  1.1.6 Examination & Digitization ............................. 11

1.2 Overview of Computational Pathology ..................... 15
  1.2.1 Terminology .............................................. 15
  1.2.2 High-Level Tasks ....................................... 17
  1.2.3 Machine Learning Strategies ........................... 20
  1.2.4 Pre-processing .......................................... 20
2 TIAToolbox as an End-to-End Library for Advanced Tissue Image Analytics

2.1 Introduction ................................................. 41
2.2 Methods ................................................... 48
  2.2.1 Reading WSI Data ..................................... 48
  2.2.2 Virtual Whole Slide Image Pyramid ................. 51
  2.2.3 Metadata .............................................. 52
  2.2.4 Tissue Masking ....................................... 52
  2.2.5 Patch Extraction ...................................... 53
2.3 Stain Normalization & Augmentation ....................... 54
2.4 Models .................................................... 56
  2.4.1 API for Models ....................................... 57
  2.4.2 Patch Classification .................................. 59
  2.4.3 Semantic Segmentation ............................... 62
  2.4.4 Nuclear Instance Segmentation and Classification .. 63
  2.4.5 Customizing Models .................................. 65
  2.4.6 Deep Feature Extraction .............................. 65
  2.4.7 Visualization ......................................... 66
2.5 Ethical Approvals for Datasets ............................. 67
2.6 Results .................................................... 67
  2.6.1 Patch Aggregator: Predicting the Status of Molecular Pathways and Mutations using Patch-level Predictions .. 69
2.6.2 Graph Aggregator: Predicting HER2 Status using SlideGraph+ .............................................. 72

2.7 Discussion .............................................. 75

2.8 Data Availability ........................................... 79

2.9 Code Availability ........................................ 80

2.10 Chapter Summary ........................................... 81

3 Whole Slide Image Conversion 83

3.1 Whole Slide Image Formats ................................. 83

3.1.1 Requirements & Use Cases ............................. 84

3.1.2 Image Formats: Containers & Codecs ................ 86

3.1.3 Tiles: Random Spatial Access ......................... 87

3.1.4 Pyramids: Multiple Resolutions ....................... 88

3.1.5 WSI Formats in Use Today ............................ 89

3.2 Why Convert Whole Slide Images? ........................ 93

3.2.1 Reduction of Complexity Through the Creation of Homogeneous Datasets ......................... 94

3.2.2 Predictable Performance Characteristics .............. 95

3.2.3 Trade-Off Between Disk Space and Speed .......... 95

3.2.4 Transitioning to Open Formats ....................... 96

3.2.5 Cloud Optimization .................................. 97

3.2.6 Clinical Import and Export ......................... 98

3.3 Challenges for WSI Conversion .......................... 98

3.4 Implementation of WSIC ................................. 101

3.4.1 Lossless Repackaging ................................. 103

3.5 Conversion Benchmarking ................................. 104

3.5.1 Bulk Repackaging of TCGA SVS Slides ... 105

3.6 Conversion Benchmarking Results ....................... 106

3.7 The Impact of Image Compression on Deep Learning Inference 107

3.7.1 Method ............................................ 109
4 Leveraging Alternative Representations of Histology Images 117

4.1 Leveraging Self-Supervised Representations 117
  4.1.1 The Proposed Method 120
  4.1.2 Evaluating the Learnt SwAV Representation 122
  4.1.3 Nuclear Composition Prediction 124
  4.1.4 Results & Discussion 128

4.2 Wavelet Compressed Representations for Large Patches 132
  4.2.1 Methods 136
  4.2.2 Experimental Setup 140
  4.2.3 Results 145

4.3 Chapter Summary 150

5 Annotating Histology Images 153

5.1 Representing & Storing Annotations 155
  5.1.1 Implementation in TIAToolbox 160

5.2 Querying for Annotations 161
  5.2.1 Performance & Accelerating Queries 162

5.3 Web-Based Tools for Annotation & Visualization 169
  5.3.1 Salsa: Web-Based Collaborative Annotation 169
  5.3.2 Aura: Heatmap Visualization 173
  5.3.3 Tango: Rapid Annotation for the Lysto Challenge 175

5.4 Chapter Summary 177

6 Conclusions 178

6.1 TIAToolbox 179

6.2 Whole Slide Image Conversion 182

6.3 Leveraging Alternative Representations of Histology Images 185
This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. I declare that, except where acknowledged, the material presented in this thesis is my own work, and has not been previously submitted for obtaining an academic degree.

Johnathan Pocock

July 2023
Abstract

Research presented in this thesis is aimed at making significant contributions in the field of computational pathology through the development of TIAToolbox, an open-source comprehensive Python library for computational pathology image analysis. The toolbox provides functionalities for tasks such as WSI reading, patch extraction, model inference, and visualization, with pre-trained models available for patch classification, semantic segmentation, and nucleus instance segmentation and classification. Additionally, the thesis explores the utility of alternative representations, including self-supervised and pooled multi-level wavelet representations, for histology image analysis. Moreover, the research addresses the challenges in annotation storage and querying, with the development of efficient storage systems capable of handling millions of annotations. These tools and techniques contribute to advancing computational pathology and provide valuable resources for future research, enabling more accurate and efficient analysis of histopathology images, thus having the potential to improve outcomes for patients.
I would like to thank my supervisor Professor Nasir Rajpoot for giving me the incredible opportunity to undertake this PhD. It was his inspiring class on computational pathology during my undergraduate studies that sparked my passion for this field. His deep knowledge and valuable insights in the field of Computational Pathology have been instrumental in shaping my research. I am especially grateful for his exceptional editing feedback. Despite his many commitments as the head of a research centre, Nasir has always dedicated his time and support. I am forever thankful for his mentorship, kindness, and ability to find perfect titles and acronyms for our projects.

I am indebted to my co-supervisor Dr Fayyaz Minhas. Beyond his remarkable expertise in bioinformatics, he has been a constant source of support and reassurance throughout this journey. I have greatly enjoyed our coffee chats, where we not only discussed research but also exchanged book recommendations and explored common interests. His friendship and guidance have been invaluable.
I am also grateful to Dr Shan E Ahmed Raza, who has been an essential collaborator in developing TIAToolbox. His consistent participation and dedication, such as conducting our weekly meetings and providing valuable insights, have significantly contributed to our success as a team.

During my PhD, Sharon Howard, the postgraduate research manager, and Maria Ferreiro, the department resources manager have been particularly helpful. Sharon’s kindness and expertise in university processes and problem-solving abilities, and Maria’s support in handling administrative tasks, have facilitated the smooth execution of my research.

Special thanks to my dear friend Dr Sam Cooper, who provided advice, and reassurance throughout my PhD. Sam has remained a constant source of inspiration and unwavering support. Our conversations gave me a valuable perspective outside my immediate research environment.

Lastly, I would like to thank my friend Alex Pappas for being my personal cheerleader and providing encouragement through the last two years of my PhD journey. His moral support and friendship were deeply appreciated and instrumental in making this journey memorable and successful.

Throughout my PhD journey, I have faced numerous obstacles. Despite these challenges, I am proud to have cultivated ‘grit’ with the support of my peers, friends, and supervisors. As Angela Lee Duckworth wrote, ‘Grit is sticking with your future day in, day out and not just for the week, not just for the month, but for years.’
# Abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 3′-Diaminobenzidine</td>
<td>DAB</td>
</tr>
<tr>
<td>Application Programming Interface</td>
<td>API</td>
</tr>
<tr>
<td>Best Alignment Metric</td>
<td>BAM</td>
</tr>
<tr>
<td>Boolean satisfiability problem</td>
<td>SAT</td>
</tr>
<tr>
<td>Cellular Composition Regression</td>
<td>CellCoRe</td>
</tr>
<tr>
<td>Class Activation Map</td>
<td>CAM</td>
</tr>
<tr>
<td>Clustering-constrained Attention Multiple Instance Learning</td>
<td>CLAM</td>
</tr>
<tr>
<td>Colorectal Adenocarcinoma</td>
<td>CRA</td>
</tr>
<tr>
<td>Computational Pathology</td>
<td>CPath</td>
</tr>
<tr>
<td>Command-line Interface</td>
<td>CLI</td>
</tr>
<tr>
<td>Computed Tomography</td>
<td>CT</td>
</tr>
<tr>
<td>Convolutional Neural Network</td>
<td>CNN</td>
</tr>
<tr>
<td>Central Processing Unit</td>
<td>CPU</td>
</tr>
<tr>
<td>Deoxyribonucleic acid</td>
<td>DNA</td>
</tr>
<tr>
<td>Digital Imaging and Communications in Medicine</td>
<td>DICOM</td>
</tr>
<tr>
<td>Digital Pathology</td>
<td>DPath</td>
</tr>
<tr>
<td>Discrete Cosine Transform</td>
<td>DCT</td>
</tr>
<tr>
<td>Term</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Discrete Wavelet Transform</td>
<td>DWT</td>
</tr>
<tr>
<td>Domain Specific Language</td>
<td>DSL</td>
</tr>
<tr>
<td>Exchangeable image file format</td>
<td>Exif</td>
</tr>
<tr>
<td>Formalin-fixed Paraffin-Embedded</td>
<td>FFPE</td>
</tr>
<tr>
<td>General Electric</td>
<td>GE</td>
</tr>
<tr>
<td>Global Interpreter Lock</td>
<td>GIL</td>
</tr>
<tr>
<td>GNU General Public License</td>
<td>GPL</td>
</tr>
<tr>
<td>Graph Neural Network</td>
<td>GNN</td>
</tr>
<tr>
<td>Graphical User Interface</td>
<td>GUI</td>
</tr>
<tr>
<td>Graphics Processing Unit</td>
<td>GPU</td>
</tr>
<tr>
<td>Grey-Level Co-Occurrence Matrix</td>
<td>GLCM</td>
</tr>
<tr>
<td>Haematoxylin and Eosin</td>
<td>H&amp;E</td>
</tr>
<tr>
<td>High-high</td>
<td>HH</td>
</tr>
<tr>
<td>High-low</td>
<td>HL</td>
</tr>
<tr>
<td>Image File Directory</td>
<td>IFD</td>
</tr>
<tr>
<td>Immunohistochemistry</td>
<td>IHC</td>
</tr>
<tr>
<td>Information Object Definition</td>
<td>IOD</td>
</tr>
<tr>
<td>Input / Output</td>
<td>I/O</td>
</tr>
<tr>
<td>Java Advanced Imaging</td>
<td>JAI</td>
</tr>
<tr>
<td>JavaScript Object Notation</td>
<td>JSON</td>
</tr>
<tr>
<td>Joint Photographic Experts Group</td>
<td>JPEG</td>
</tr>
<tr>
<td>JPEG File Interchange Format</td>
<td>JFIF</td>
</tr>
<tr>
<td>Le Gall-Tabatabai</td>
<td>LGT</td>
</tr>
<tr>
<td>Low-high</td>
<td>LH</td>
</tr>
<tr>
<td>Low-low</td>
<td>LL</td>
</tr>
<tr>
<td>Lung Adenocarcinoma</td>
<td>LUAD</td>
</tr>
<tr>
<td>Term</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Lung Squamous Cell Carcinoma</td>
<td>LUSC</td>
</tr>
<tr>
<td>Mean Absolute Error</td>
<td>MAE</td>
</tr>
<tr>
<td>Mean Squared Error</td>
<td>MSE</td>
</tr>
<tr>
<td>Microns-per-pixel</td>
<td>MPP</td>
</tr>
<tr>
<td>Multi-layer Perceptron</td>
<td>MLP</td>
</tr>
<tr>
<td>Multiple Instance Learning</td>
<td>MIL</td>
</tr>
<tr>
<td>Neutral Buffered Formalin</td>
<td>NBF</td>
</tr>
<tr>
<td>Next-Generation File Format</td>
<td>NGFF</td>
</tr>
<tr>
<td>Non-Small-Cell Lung Carcinoma</td>
<td>NSCLC</td>
</tr>
<tr>
<td>Office for National Statistics</td>
<td>ONS</td>
</tr>
<tr>
<td>Open Microscopy Environment</td>
<td>OME</td>
</tr>
<tr>
<td>Ordinary Least Squares</td>
<td>OLS</td>
</tr>
<tr>
<td>Peak Signal to Noise Ratio</td>
<td>PSNR</td>
</tr>
<tr>
<td>Picture archiving and communication system</td>
<td>PACS</td>
</tr>
<tr>
<td>Potential of Hydrogen</td>
<td>pH</td>
</tr>
<tr>
<td>Python Package Index</td>
<td>PyPI</td>
</tr>
<tr>
<td>Red, Green, and Blue</td>
<td>RGB</td>
</tr>
<tr>
<td>Region of Interest</td>
<td>ROI</td>
</tr>
<tr>
<td>Residual Network</td>
<td>ResNet</td>
</tr>
<tr>
<td>Single-reader multiple-writers</td>
<td>SWMR</td>
</tr>
<tr>
<td>Small-Cell Lung Carcinoma</td>
<td>SCLC</td>
</tr>
<tr>
<td>Structural Similarity Index Measure</td>
<td>SSIM</td>
</tr>
<tr>
<td>Structured Query Language</td>
<td>SQL</td>
</tr>
<tr>
<td>Swapping Assignments Between Views</td>
<td>SwAV</td>
</tr>
<tr>
<td>Tagged Image File Format</td>
<td>TIFF</td>
</tr>
<tr>
<td>The Cancer Genome Atlas</td>
<td>TCGA</td>
</tr>
<tr>
<td>Term</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Tissue Image Analytics</td>
<td>TIA</td>
</tr>
<tr>
<td>Tumour-Infiltrating Lymphocyte</td>
<td>TIL</td>
</tr>
<tr>
<td>Variational Autoencoder</td>
<td>VAE</td>
</tr>
<tr>
<td>Visible Light</td>
<td>VL</td>
</tr>
<tr>
<td>Well-Known Binary</td>
<td>WKB</td>
</tr>
<tr>
<td>Well-Known Text</td>
<td>WKT</td>
</tr>
<tr>
<td>Whole Slide Image</td>
<td>WSI</td>
</tr>
<tr>
<td>Whole Slide Image Converter</td>
<td>WSIC</td>
</tr>
</tbody>
</table>
List of Figures

1.1.1 Flow chart illustrating the steps of a typical cancer diagnosis pipeline. ......................... 6
1.1.2 Diagram of a tissue sample being viewed with a microscope. ........................................ 7
1.1.3 Photograph of an FFPE tissue block. .......... 8
1.1.4 Photographs of a tissue block being cut and mounted on a slide. ................................. 9
1.1.5 Photograph showing baths containing stains. .... 12
1.2.1 CPath pipeline block diagram ....................... 19
2.2.1 Illustration of read_rect and read_bounds. ........... 50
2.4.1 TIAToolbox framework diagram. .......... 58
2.4.2 TIAToolbox semantic segmentation. ........ 60
2.6.1 TIAToolbox SlideGraph and IDaRS pipeline diagram. 68
2.6.2 TIAToolbox capabilities ......................... 74
3.1.1 Venn diagram of primary use cases for WSIs .......... 86
3.1.2 Illustration of an image pyramid. .................. 90
3.4.1 Diagram illustrating the SWMR pattern used by WSIC. 102
3.6.1 Bar chart of conversion speed across the compared
open-source CLI conversion tools. .......................... 105
3.7.1 PSNR and compression ratio vs pixel-wise F1 coeffi-
cient for nucleus segmentation. ............................... 112
3.8.1 Plot of PSNR against the percentage of detected cell
type. .......................................................... 114
3.8.2 Plot of compression ratio against percentage of de-
tected cell type. ................................................ 115

4.1.1 Concept diagram for cell count regression (CellCoRe). 119
4.1.2 UMAP of SwAV embeddings for tissue classification. 123
4.1.3 Lone normal patch in a cluster of mucus ............... 124
4.1.4 Scatter plots of UMap embedded SwAV features for
the PanNuke dataset. ........................................... 126
4.1.5 Hexbin plots for cell count prediction. .................... 129
4.1.6 Ground truth labels and predicted cell type counts
for patches from the PanNuke-Breast dataset. ............ 130
4.1.7 Parameter count versus Pearson’s r for four-class cel-
lular composition on PanNuke-Breast. ...................... 131
4.2.1 Concept diagram of patch-based method vs proposed
wavelet decompose representation. .......................... 135
4.2.2 Pooled multi-level discrete wavelet transform diagram. 138
4.2.3 Frequency response maps after multi-level DWT and
max pooling. ...................................................... 139
4.2.4 Score-CAM activation map of low-grade CRA ........ 151
4.2.5 Score-CAM activation map of high-grade CRA . . . . 151
5.1.1 Plot of size vs time for compressing WKB annotation
geometries . . . . . . . . . . . . . . . . . . . . . . . . 160
5.2.1 Bar plot of time to appends annotation for Dictionary
Store and SQLiteStore. . . . . . . . . . . . . . . . . 163
5.2.2 Bar plot of polygon intersection query time. . . . . . 164
5.2.3 Bar plots showing time for querying millions of point
annotations and querying with a predicate using an
index. . . . . . . . . . . . . . . . . . . . . . . . . . . 164
5.2.4 Time to find overlapping polygon annotations. . . . . 168
5.2.5 Line plot of neighbourhood query time with SQLite
Store.nquery. . . . . . . . . . . . . . . . . . . . . . 168
5.3.1 Collaborative editing with Salsa. . . . . . . . . . . . . 170
5.3.2 Salsa user interface. . . . . . . . . . . . . . . . . . . . 172
5.3.3 Aura user interface showing a heatmap. . . . . . . . . 174
5.3.4 Tango mobile user interface for point annotation. . . 176


List of Tables

2.7.1 Comparison of features available in different histopathology image analysis focused software packages. . . . . . 76

4.1.1 Test metrics for cell count regression models. . . . . . 131
4.2.1 Accuracy and input representation comparison of models on the NCT-CRC-HE-100K (Kather 100K) dataset. 148
4.2.2 Input Representations with accuracy and weighted accuracy for colorectal adenocarcinoma grading. . . . 149
4.2.3 Lung subtyping (CPM dataset) results. . . . . . . . . . . 150

B.1 Detailed metrics of Models for patch prediction. . . . 227
B.2 Patch classification performance of models provided by TIAToolbox. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 228
B.3 Semantic segmentation performance (Sørensen-Dice score) on the Breast Cancer Semantic Segmentation (BCSS) dataset. . . . . . . . . . . . . . . . . . . . . . . . 228
B.4 Performance of IDaRS provided as part of TIAToolbox, compared to the original implementation. . . . . 229
B.5 Performance of SlideGraph+ in TIAToolbox. . . . . . 229
B.6 Detailed metrics of Models for semantic segmentation. 229
B.7 Detailed metrics of Models for nucleus segmentation
   or classification. . . . . . . . . . . . . . . . . . . . . . 229
B.8 Whole slide classification with SlideGraph and IDaRS. 230
List of Listings

A.1 TIAToolbox patch extraction, reduction in lines of code. 222
A.2 Synchronous slide and mask reading with TIAToolbox. 223
A.3 Prediction of pre-extracted patches with TIAToolbox. 223
A.4 WSI patch prediction with TIAToolbox. 223
A.5 WSI semantic segmentation with TIAToolbox. 224
A.6 WSI nucleus instance segmentation with TIAToolbox 224
A.7 Using pre-trained weights for models in TIAToolbox. 224
A.8 Graph visualization with TIAToolbox. 224
A.9 Web tile server with TIAToolbox. 225
A.10 SQLiteStore usage examples. 225
A.11 IDaRS inference using TIAToolbox. 226
Chapter 1

Introduction

Half of all people in the United Kingdom (UK) born after 1960 will be diagnosed with cancer in their lifetime [1], [2]. Disease is an inevitable part of human existence. When disease does occur, it is imperative to have accurate diagnostic tools at our disposal. Histopathology, the examination of tissue samples for diagnosing diseases, is considered the gold standard for the diagnosis of cancers and many other diseases due to its accuracy and reliability. With cancer being the cause of 24% of deaths in the UK [3] and a leading cause of death worldwide [4], accurate diagnostic tools such as histopathology are crucial for timely and effective treatment. The National Institute for Health and Care Excellence (NICE) estimate that 1.2 billion pathology tests are conducted annually in England alone and that around 95% of clinical pathways rely on pathology services [5].

Despite its importance, there is a notable shortage of pathologists available to perform this work, with only three per cent of
NHS trusts reporting adequate staffing to meet demand [6]. This is where the digitization of slides and the implementation of automation techniques can prove invaluable. By utilizing image processing and machine learning algorithms, such as deep neural networks, the workload of pathologists can be significantly reduced, increasing their throughput and enabling them to handle the large volume of cases more effectively. This can improve patient outcomes by accelerating the diagnostic process and identifying potential oversights.

This thesis describes the development of tools and methods to improve the effectiveness of image processing and machine learning to achieve optimal patient outcomes.

Machine learning algorithms can systematically and exhaustively analyse slide images that would be impractical to search manually. This additional layer of scrutiny can help ensure that no potential issues are overlooked and that patients receive the highest standard of care.

However, the increasing amount of automation across many industries has increased concerns about replacement by machines. So much so that the Office for National Statistics (ONS) now hosts a web page to calculate your risk of replacement by automation [7]. Furthermore, much has been written on variants of the question, ‘Can an algorithm replace pathologists?’ [8], [9], suggesting a fear of replacement is in the zeitgeist for pathologists. Meanwhile, patients may be uncomfortable with the idea of no human involvement in the diagnostic process, known as an ‘algorithm aversion’. Experiments have demonstrated that humans prefer forecasts made
by other humans to those made by statistical models, even when provided with evidence that shows the human to be less accurate [10].

However, it is unlikely that pathologists will be automated out of a job anytime soon [9]. Although the ONS does not have a specific ‘pathologist’ category, a medical practitioner is scored as the least likely occupation to be automated. The consensus appears that digital and computational pathology are far more likely to augment the work of humans, much like other technologies have already done in other areas such as radiology [8], [9]. For instance, it can automate tedious tasks such as counting cells within a slide area and quantifying the intensity of a stain used to detect the presence of a specific protein [11].

1.1 The Histopathology Pipeline

Before discussing digital and computational pathology, it’s crucial to understand how tissue is obtained and processed before it can be digitized. This digitization is where the digital and computational workflows diverge from traditional pathology. The quality of the tissue samples and the methods used to process them can greatly affect the accuracy and reliability of the resulting digital images. By thoroughly understanding these processes, researchers without a rigorous pathology background can better interpret the data and make more informed decisions when analysing histopathological images.
1.1.1 Clinical Terms

The terms histology, pathology, and histopathology are easily confused and may be used ambiguously or interchangeably in some contexts. Before detailing the steps in a histopathology pipeline, it may also be instructive for the reader to have these and other key clinical terms defined:

**Histology:** The study of the structure and function of microscopic tissues in living organisms.

**Pathology:** The study of disease, its causes, development, and consequences. The practice of pathology involves all bodily specimens including tissue and histological examination.

**Histopathology:** The study of abnormal tissue resulting from disease.

**Gross Examination** (‘Grossing’) Examination of tissue with the naked eye.

**Clinical Investigation** A clinical test or investigation offered to or carried out on a living organism. It may include a procedure such as blood tests for specific antibodies, scans, or physical examinations for specific diseases.

**Procedure:** A medical intervention that involves using instruments or devices to diagnose or treat a condition. Examples of procedures include biopsies, endoscopies, and surgeries such as resections.
(Physical) Examination The process of evaluating objective anatomic findings through the use of observation, palpation (touching), percussion, and auscultation (listening). It is also known as a physical diagnosis.

1.1.2 High-Level Overview of the Pathology Pipeline

A patient may initially present with symptoms, for example, the appearance of a new lump; a doctor will take a history of the patient’s symptoms and perform a physical examination. In addition to diagnosing patients with symptoms, screening through histopathology is also an important tool for detecting cancer early on. The doctor may also order investigations such as blood tests and imaging studies (e.g., X-rays, CT scans), including procedures such as biopsies to help make a diagnosis. Next, the tissue sample is processed before being examined by a pathologist, who looks for abnormal cells or tissues that may indicate cancer. Finally, a report is written by the pathologist that includes the diagnosis and any other relevant information.

1.1.3 Fixing & Embedding

In a typical microscopy setup, light is emitted from a source underneath the tissue, before travelling through a glass slide, the tissue sample, a protective coverslip, and multiple lenses before reaching the eye of an observing pathologist. In order to transmit light, the tissue must be extremely thin. Samples are sliced to a width of 4–5 µm, less than the width of a human red blood cell (~7 µm). To
create this semi-transparent thin slice, the tissue must go through several processing steps. Tissue must be ‘fixed’ and embedded in wax, or sometimes frozen instead, to preserve the tissue and make it hard enough to cut. It must then be sliced very thinly to allow transmission of light before being stained to highlight structures and reveal the presence of compounds such as deoxyribonucleic acid (DNA) and proteins.

The process of fixing is performed with a 4% aqueous solution of formaldehyde (formalin) mixed with a buffer solution to create a mixture of 10% neutral (pH 7) buffered formalin (NBF). The buffer in NBF prevents pH-sensitive changes to the tissue. The formaldehyde causes cross-linking of proteins, kills bacteria, and denatures proteins such as enzymes in the tissue which hardens the tissue and
helps prevent decomposition.

In order to embed the tissue in paraffin wax, the tissue must first be dehydrated. This is because paraffin is hydrophobic and the NBF solution is 90% distilled water. Therefore, the NBF must be removed or replaced with something anhydrous that will mix with paraffin. This dehydration process is performed by immersing the specimen in increasingly strong concentrations of ethanol, typically from 70% through to 100% concentration. However, the ethanol must now also be ‘cleared’ because paraffin and ethanol are also largely immiscible. Clearing not only removes the ethanol but also some fat from the tissue which can prevent the paraffin from penetrating and leaves the tissue optically clear. A popular clearing agent is xylene.

To prepare the tissue for embedding, the cleared tissue sample is carefully positioned inside a mould before injecting warm liquid
paraffin. A plastic cassette is placed on top, and the entire setup is moved onto a cold plate to harden. This results in a tissue ‘block’ that is formalin-fixed and paraffin-embedded (FFPE). This common processing method takes several hours or days [12]. However, there are other processing procedures available, such as flash freezing for emergency pathology to identify a tumour during surgery [12].

This long multi-stage process of fixing, dehydrating, clearing and embedding is designed to preserve the tissue as much as possible. However, tissue removed from a patient will immediately degrade due to a variety of factors, including loss of blood supply. As such, it is also important to begin fixing tissue as soon as possible after extraction. The strong chemicals and dehydration will cause some deformation and warping to the tissue, which is visible in the final slide. This is unavoidable when preserving tissue and preparing it for examination or digitization.

Figure 1.1.3: Photograph of an FFPE tissue block in a white plastic cassette after the sample has been fixed, dehydrated, cleared, and embedded in paraffin.
1.1.4 Sectioning

Before the embedded tissue can be examined, a thin slice must be extracted and stained to enhance contrast and highlight structures. Although the tissue block is hard when cooled, it is soft enough to easily cut with a sharp blade. A specialized cutting device, a microtome, creates wafer-thin sections approximately 4–5 µm thick. Wafers become crinkled during cutting and must be smoothed out again by floating them on a warmed water bath. However, this process is not perfect at smoothing out a wafer section, and some folds will remain. These will be visible in the final tissue slide. Furthermore, tiny tears in the tissue, known as ‘knife chatter’, can occur during cutting. This may be caused by over-processing or overly dehydrated tissue. Chatter can also be caused by a blunt blade or poor technique.
Artefacts such as folds or chatter in an image may potentially obscure or distort features in the slide image, which deep learning models may pick up on when training. However, for a large enough dataset, the model will likely become robust to this variation by incidental inclusion of images which both do and do not feature these artefacts. However, as with most artefacts, there is also the potential for batch effects to occur, e.g., in a multi-centre study. Sections from a specific centre may have a consistent and thus detectable pattern in knife chatter. A model may be able to pick up on this pattern and use it to linearly separate patient cohorts by the centre rather than relying on biologically prognostic relevant features in the image.

1.1.5 Staining

Tissue samples are stained before histological examination to enhance the visualization of cellular structures and metabolic processes. Staining techniques can penetrate cell walls and highlight specific cellular components, allowing for more detailed analysis. The most common staining method combines two stains: haematoxylin and eosin (H&E). When H&E reacts with a tissue sample, nuclei are stained blue, whereas the cytoplasm and extracellular matrix have varying degrees of pink staining.

Haematoxylin, extracted from the wood of the logwood tree (Haematoxylum campechianum) [13], has a deep blue-purple colour. When combined with a mordant\(^1\) it forms a cationic (positively charged) basic stain that binds to nucleic acids such as DNA [13]. As haematoxylin is a chemical that fixes or intensifies a stain.

\(^1\)A mordant is a chemical that fixes or intensifies a stain.
toxylin is naturally derived rather than synthesized, there is significant variation between batches, which can affect staining intensity\cite{13}.

Eosin is pink and stains proteins nonspecifically. Eosin refers to two fluorescent acidic compounds, eosin Y and eosin B. However, eosin Y is the most commonly used in histology [13]. Eosin is an anionic (negatively charged) acidic dye that can bind to positively charged components in tissue, such as amino groups in proteins found in the cytoplasm, extracellular matrix, and collagen.

Notably, staining can vary due to the exact procedure, which is also not standardized across pathology laboratories. Although staining within a lab is now commonly consistent due to being performed by machines with a robotic arm, manual staining still occurs and is one source of variation in the staining intensity. Furthermore, since haematoxylin is not a synthesized chemical but a naturally derived compound, there is some natural variation in quality, purity, and strength. There has been some debate about whether staining quality and intensity is a problem for deep learning models. Much work has been published on digitally separating, normalizing \cite{14}, \cite{15} or augmenting \cite{16} the staining of H&E images when training models.

1.1.6 Examination & Digitization

‘On-glass’ examination refers to the traditional method of examining histological slides using a conventional light microscope. In this approach, the pathologist physically reviews the slides by placing
them under a microscope and directly looking at the tissue samples on a glass slide.

First, the processed slide is placed onto the stage of a microscope. The pathologist adjusts the objective lens (the lens closest to the slide) to the lowest magnification, which provides a broad overview of the tissue. This initial low-magnification examination helps the pathologist become familiar with the overall structure of the tissue and locate any interesting areas, also known as regions of interest (ROIs), for further investigation.

Next, the pathologist gradually increases the magnification by rotating the objective lens to higher powers, such as 10× or 40×. These higher magnifications allow a closer look at cells and tissues at a microscopic level.
As the pathologist observes the slide, they look for any abnormalities or changes from typical tissue characteristics. This might include alterations in cell structure, cell arrangement, the presence of any foreign materials, or ink markings. Tumour cells ‘in the margin’, close to ink, may indicate that not all the tumour was removed.

Many hospitals are currently shifting from the conventional on-glass pathology workflow, which involves examining physical glass slides under a microscope, to a modern digital process. This new process entails scanning each slide after preparation and saving the resulting image to a picture archiving and communications (PACS) system. The digitized slide images are called whole slide images (WSIs) and are examined on a computer monitor instead of under a microscope. A WSI may be very large, typically tens of thousands of pixels in width and height. This large image size makes capturing, storing, viewing, and processing these images challenging. As a result of their large dimensions, WSIs also occupy a large amount of disk space, typically multiple gigabytes, even when using compression. Furthermore, most typical workstation PCs will not have enough working memory (RAM) to store the entire uncompressed image. As uncompressed pixel data is required in memory for viewing and manipulation, image regions can be efficiently read from the disk as required to work around this constraint. Lastly, image processing operations, such as rescaling image data for viewing, can be very computationally intensive for large images. This can be addressed by only processing the regions of the image currently required or by precomputing different resolutions.
However, this digital workflow does have several advantages over the traditional on-glass examination. Digitizing slides allows pathologists to access and analyze WSIs remotely, reducing glass slides’ physical handling, shipping, and storage. This may enable faster and more efficient diagnosis. Another advantage is increased flexibility in the viewing of slides. For example, it is easier for multiple people to simultaneously look at a digitized slide. Slides may also be transferred over the internet to other pathologists, for example, to get the opinion of an expert on the other side of the globe. Lastly, having slides in a digitized format enables a wide range of automated analysis, either by hand-crafted statistical algorithms or via data-driven machine learning models. Such automated analysis can potentially output robust, reproducible, and quantifiable metrics of clinical significance. Furthermore, a machine can perform this processing quickly, systematically and rigorously, which may free pathologists time to perform higher-level decision-making tasks and increase overall throughput for a pathology lab.

One example where automation is beneficial for cell quantification. This task is tedious and currently done by hand, often using a small representative region of the slide. It also has high inter- and intra-observer variability [17], [18]. A computational method could provide more accurate and consistent results, with the added benefit of analysing the entire slide instead of a small sample region.
1.2 Overview of Computational Pathology

Computational pathology has the potential to be applied to many parts of the pathology pipeline, from real-time analysis and diagnostics during sample collection to triaging of cases before scanning, analysis of digitized slides, and electronic reporting [19]. Digitized slide analyses have immense potential for throughput optimization and enhanced systematic and consistent pattern recognition, especially through automation. Computational pathology plays a significant analytical role in this field, as evidenced by much prior work and the focus of this thesis. This section gives an overview of existing methods and research informing this thesis.

1.2.1 Terminology

It may be helpful to define some additional terms that will occur often in this section and the thesis.

**Natural Images:** Images commonly encountered in everyday life, such as photographs, scanned documents, paintings, and drawings. **Visual Field:** An image region extracted from a WSI, typically similar in size to natural images.

**Patch:** A small sub-region of a whole slide image. Sets of patches are typically square and homogeneous in width and height, as this is required for many machine-learning methods. Patches are typically smaller than visual fields. *(Machine Learning)*

**Model:** A representation of a process that can make predictions or decisions based on input data. Model is an abstract term...
typically taken to be a set of adjustable (learnable) parameters combined with an algorithm to transform data from one form to another.

**Feature:** Some measurable property, characteristic or pattern. Features may include statistics such as a count, variance, or histogram of values. They may also include labels predicted by a neural network. Handcrafted/Classical Features: Manually engineered features that are designed to capture specific information or patterns in the data that are relevant to achieving a task.

**Deep Learning (DL):** A subset of machine learning that involves training artificial neural networks with multiple layers to solve complex problems. Deep learning typically refers to learning where features are learnt by the model directly from the data and not explicitly defined.

**Data Augmentation:** Transformations to data which do not alter the semantic meaning of the data. For example, adding random noise to an image or applying a random flip.

**Feature engineering:** The process of selecting and transforming the most relevant and informative features obtained from raw data to improve the performance of machine learning algorithms.
1.2.2 High-Level Tasks

Computational pathology may have many different high-level goals or tasks that reflect the diverse nature of a pathologist’s work. These include providing decision support metrics and making predictions such as for patient survival, mutation status, treatment response, and identifying potential regions of interest. These high-level tasks may be performed at a slide level, with an entire WSI or large enough visual field, or by combining predictions across several slides within a patient case.

Decision support can be provided to pathologists by calculating consistent, quantitative metrics with diagnostic or prognostic value. It is known that there is high inter- and intra-observer variability [17], [18] for tedious pathology tasks such as cell counting. Cell counting may be required when assessing nuclei’s staining intensity and frequency in immunohistochemically (IHC) stained samples [17], [18] to quantify the expression of a specific gene. An automated computational method of counting stained cells and quantifying the intensity of staining could provide more accurate and consistent results, with the benefit of being able to systematically analyse the entire slide instead of only a small representative sample region, as is currently done by human observers due to time and workforce constraints. Using such systematic exhaustive methods to detect and quantify biologically significant features could be highly effective in triaging cases or bringing attention to salient regions, such as those containing rare and easily overlooked patterns or like dividing mitotic cells,
lone tumour cells, or potential false positives such as cells which mimic the appearance of tumour cells.

Another useful but difficult high-level task is predicting patient survival. However, many factors that can influence a patient’s outcome need to be accounted for. Additionally, the data may be right-censored, meaning that the follow-up of a patient is lost at some point. Furthermore, finding a strong prognostic signal within a large image is sometimes challenging. There may be only a very small region of tumour cells [20] or a subtle visible phenotypical alteration (change in visual appearance) caused by genetic variations such as microsatellite instability (MSI) in colon cancer [21].

Prediction of specific mutations in genes such as TP53 (Tumour protein 53), the most frequently mutated gene (> 50%) in human cancer [22], [23], from only routine H&E stained images [24] are useful high-level prognostic indicators which may alter patient treatment such as the kind of therapy offered such as immunotherapy or hormone therapy. Accurate enough correlation of gene expression from H&E has the potential to reduce costs by avoiding the need for additional testing or genetic sequencing [25].

Furthermore, directly predicting a patient’s likely response to an intervention such as hormone therapy is of great value. One example of this is via predicting estrogen and progesterone receptor status directly from H&E images without IHC or genomic testing [26]–[28]. The predicted receptor statuses can then be used to predict the response to hormone therapy for breast cancer. Clinicians can then tailor their treatment approach and choose the most effective
Figure 1.2.1: A block diagram showing a typical CPath pipeline with a two-stage approach where the slide is pre-processed before features are extracted, and then aggregated to make a slide-level prediction.

therapy for each patient. Predicting patient treatment response may lead to improved patient outcomes and a more personalized approach to cancer treatment.

Building an algorithm or pipeline that can reliably perform these high-level prediction tasks is often done by first detecting local-level features such as the tissue type, nucleus type, and shape or identifying glands and their morphology. After extracting these foundational biologically significant features, an aggregation method (Section 1.2.7) can be used on top of these features to create a high-level estimator which operates at a slide or case level.
1.2.3 Machine Learning Strategies

Two main methods of applying machine learning to histology images are the two-stage method and the end-to-end learning strategy. The former involves extracting local features, selecting relevant ones, and aggregating them to build up a representation of the whole image. The latter involves training a model to perform a complete task from input to output without human intervention. The two-stage approach is more common, as it allows for some level of explainability, scalability, and flexibility. However, the end-to-end approach can automate feature extraction and selection, resulting in improved performance and reduced development time. Nonetheless, it can be computationally expensive, require large datasets, and may lack interpretability [29].

1.2.4 Pre-processing

Before extracting features from a WSI, some pre-processing may be required.

Tissue Masking

Tissue masking is an initial step in the image processing pipeline that generates a segmentation mask. This mask separates diagnostic tissue areas from non-diagnostic slide backgrounds. Generation of this mask is designed to speed up downstream analysis steps by quickly eliminating image regions that do not need to be analysed because they offer no diagnostic or prognostic value.
A low-resolution copy of the WSI is typically used to generate the mask, derived from a high level of the WSI pyramid. Image processing commonly uses binary threshold filters like Otsu’s method [30], which can automatically find a good threshold to create a mask.

Post-processing of the mask may be applied using morphological operations to smooth the mask, fill small holes in tissue regions, or ignore very small islands of tissue region. More advanced methods, such as a CNN or random forest, may also be used. However, they require a large dataset to train and are slower than a simple threshold.

Additionally, masks may not be limited to two classes for detecting foreground versus background but aim to detect other areas that may need to be excluded, such as pen markings, blurry regions, and large tissue folds.

**Patch Extraction**

After identifying regions of tissue within the images, a common step is to extract patches within the tissue region. Patches may have some overlap to avoid border effects during feature extraction. Additionally, when reaching the edges of the WSI, patches may be dropped or padded to keep the set of patches homogeneous in size. This is important as many model training algorithms and frameworks, such as for training CNNs, require batches of examples which are all the same size.
Stain Normalisation & Augmentation

Changes in staining protocol, batch effects for staining, and differences in colour rendering between scanners can lead to large variations in the appearance of slides. Such variances may introduce batch effects, which could provide a false signal for machine learning models [31]. Training on a large multi-centre dataset may help models to become robust to such variation. However, this may not be practical or possible in all cases, especially for rare conditions with less data available. As such, stain normalisation may be applied to normalise tissue appearance across batches and sites.

When training deep learning models, it can be helpful to randomly vary the intensity of each stain present in the image. This technique, known as stain augmentation, involves introducing artificial variations during training. Stain augmentation can aid the model in becoming more robust to variance in staining intensity, making it more accurate in predicting class labels. Augmentation is a common strategy used in deep learning models to introduce additional intra-class variance in the training data and encourage invariance to certain transformations known to not significantly affect the class label. Other common augmentations include random horizontal flipping, the addition of random noise, and random redaction of image regions. Flipping is typically limited to horizontal flipping, as ‘natural images’, i.e. the kind of everyday images taken with a camera, usually have a clear global orientation. Therefore, vertical flipping does not make sense for this kind of image. However, histol-
ogy images have no global orientation, which means that additional augmentations such as vertical flips and random rotation may be applied to increase the robustness of the model.

**WSI Registration**

Image registration is the process of aligning two or more images of the same scene taken at different times, from different viewpoints, or by different modalities. It aims to find a transformation that maps one image onto the other(s) so that corresponding points or features in the images are brought into alignment. This may be useful in computational pathology to register serial slices taken from the same tissue block, which can give information at different depths. It may also align different modalities or stains, such as H&E and IHC slides.

As WSIs are typically spatially very large and may be several gigabytes, registering WSIs can be challenging due to the large amount of memory required and the large compute time for each step in iterative approaches. Additionally, the difference in appearance between modalities and the presence of artefacts such as tissue folds, staining variations, or other scanning artefacts can complicate alignment.

Rigid registration is the simplest type of image registration that involves only translation, rotation, and scaling transformations, while elastic registration allows for more complex deformations such as stretching, bending, or twisting. Elastic registration is more flexible and can handle more diverse variations in the images, but it is also
more computationally intensive and may require more specialized algorithms.

**Feature Extraction & Engineering**

Low- or local-level tasks, also referred to as feature extraction, aim to extract useful features from the image without requiring the context of an entire slide, usually operating on independent regularly sized regions referred to as patches. By working on local independent patches, features can be extracted quickly in parallel and without using much memory per patch. This makes it easier to scale the extraction of features up and down to suit the constraints of available hardware. Furthermore, local-level tasks are often selected to not depend on a larger context beyond the patch level. For example, detecting cells within a patch does not require knowing the contents of surrounding patches, even though knowing that information may provide a helpful statistical prior in some cases.

Locally extracted features may be statistical or ‘classical’ handcrafted features, such as a histogram of colours within the patch, a histogram of gradients, or a grey-level co-occurrence matrix [32]. Alternatively, features may be predictions made by a trained deep neural network. Predictions may be patch-level labels, such as the tissue type, or objects within the patch, such as annotations (labelled polygon boundaries) of all nuclei within the patch.

Machine learning models can also be used to generate features. Models may be trained as sub-tasks within a larger two-stage pipeline. They can either extract useful features directly from the image, us-
ing ‘deep’ learning models or convert basic handcrafted features into more informative ones, using ‘classic’ learning models. Convolutional neural networks (CNNs) are commonly used deep learning models which train both features and models end-to-end. Besides, classical ML models such as support vector machines (SVMs) and random forests, which take previously extracted features as input, may also be utilized. ML models are often useful for complex feature extraction tasks such as image classification and segmentation, where traditional methods may not be as effective. However, they can be computationally expensive and require large training datasets. Therefore, the choice of feature extraction method depends on the specific task and available resources.

Some common feature extraction tasks in the CPath domain are patch classification (such as coarse tissue classification, artefact detection, and blur detection), per-pixel label prediction or semantic segmentation (such as fine-grain tissue classification), object detection (such as nucleus instance segmentation or gland segmentation), and object classification (such as nucleus classification).

**Use of Classical Features**

Classical features, or handcrafted features, have been used for decades in machine learning and computational image analysis. They have been extensively studied and applied in various applications, including medical image analysis. One of the advantages of classical features is that they are more easily analyzed and explained than deep learning models, where features are learned directly from the
data. Additionally, classical features have a strong foundation in prior human knowledge and intuition, allowing them to provide a powerful discriminative signal to downstream analysis without requiring a separate model for feature extraction. This can save time and reduce the need for large amounts of data.

One such feature is the Grey-Level Co-Occurrence Matrix (GLCM), which is a matrix that represents the spatial relationship between pairs of pixels based on their grey-level values. GLCM has been used in texture analysis of natural and overhead satellite images for many years [32], and it has been shown to be effective in differentiating between different types of tissues and identifying abnormalities [33]. It continues to be used effectively in modern medical image analysis along with other hand-crafted textural features such as the discrete wavelet transform (DWT) [34].

Many handcrafted features have been created and shown to be effective on various image analysis tasks. Some common classical features include intensity-based features such as histogram, mean, variance, and skewness, as well as edge-based features such as gradient and Laplacian of Gaussian. These features have been widely used in medical image analysis. For instance, handcrafted features have been used to differentiate between different types of tissues [33] and detect objects in medical images [35].

Although deep learning models have shown remarkable performance in various applications, where features are learnt from the data rather than handcrafted, classical features still play a role in medical image analysis. They can be used to extract meaningful
information from medical images and are more amenable to explainability and analysis than deep learning models. Deep learning models are often called a ‘black box’, as the chain of causation from the input pixels to output prediction can be hard to untangle. Furthermore, classical features can be combined with deep learning models to improve the accuracy and reliability of medical image analysis systems [35].

1.2.5 Features from Classical & Deep Learning Models

In this section, we explore the use of features extracted from both classical and deep learning models in medical image analysis. We will discuss how these features can be combined to improve the accuracy and reliability of medical image analysis systems, particularly in patch prediction tasks.

Patch Prediction

Patch prediction is a common sub-task in CPath pipelines, and the output labels from a patch prediction model may be used as features for aggregation or being fed into another model. Good patch prediction is typically achieved with an end-to-end trained model, commonly a CNN. These models can predict many discrete labels, such as binary tumour versus non-tumour [36]–[38] or multi-class tissue type [33], in addition to continuous values, such as staining intensity.
Semantic Segmentation

Semantic segmentation aims to label objects within an image, often at a per-pixel level, with a class corresponding to the semantic meaning of the object or region represented by that pixel. This is a fundamental task in computer vision and has many applications, such as autonomous driving, medical imaging, and object recognition. Examples of semantic segmentation tasks in computational pathology include fine-grain tissue classification, nucleus segmentation, and gland segmentation. There are several approaches to this task, but the most common one involves using CNNs with two dominant architectures in CPath named DeepLab [39] and U-Net [40].

Instance Segmentation & Classification

One important limitation of semantic segmentation methods is their inability to distinguish touching instances within an image. Unlike semantic segmentation, instance segmentation detects objects and distinguishes between different instances of the same object. This may be achieved with various techniques, one of which is simply postprocessing the result of a semantic segmentation method, such as U-Net. A predicted semantic segmentation mask can be transformed into instances by attempting to separate touching or overlapping instances. Watershed algorithms, of which there are many variants, are widely used for separating touching objects in a semantic label map. With a flood-fill kind variant of the watershed
algorithm, semantic labels are first transformed into a distance map, where each foreground is replaced with the distance to the nearest background pixel. Next, peak values within this new map are used as locations or ‘sources’ from which multiple flood fills are performed. Where each flood fill meets another, a watershed boundary is added. However, they may not always produce a good result, for example, if there is noise in the predicted semantic labels.

**Deep Features**

Deep features, also known as neural codes, refer to the output of one or more intermediate layers of a pre-trained CNN that has been trained on a large dataset [41]. These features are a compressed representation of the input image that captures its salient characteristics. The power of deep features lies in their ability to capture high-level abstract features such as texture, shape, and colour, which are difficult to extract using traditional computer vision techniques [42]. This makes them extremely useful in a wide range of applications, such as image classification, object detection, and image retrieval [43]. By using pre-trained CNNs and extracting deep features, developers and researchers can save considerable time and computational resources that would otherwise be required to train a CNN from scratch [44].

Deep features are well known to be highly effective in a wide range of computer vision applications [43], [45].
Representation Learning

Representation learning is a machine learning technique used to create a compressed representation of data, usually as a vector of numbers that captures the distinguishing features of data points. This technique is also known as feature learning or unsupervised feature learning. There are several methods for achieving this, including generative adversarial networks (GANs) and variational autoencoders (VAEs), which both function as encoders and decoders to learn how images can be represented as a small vector of numbers that can recreate the original image.

1.2.6 Feature Engineering

After extracting features, they may be transformed into additional ones through feature engineering. This involves selecting and modifying basic features to create new, more informative ones. For example, statistical morphology features can be calculated from predicted nucleus annotation. This can be helpful since there is a known link between cell morphology, dysplasia, and malignancy.

1.2.7 Aggregation

After extracting local features from a whole slide image (WSI), these features must be combined to make an overall prediction for the WSI. This process is known as aggregation, and there are many different strategies.

One simple but effective method is a majority voting of local
patch-level predictions 2022 [46]. This method has been proven effective in predicting mutation status, such as MSI, from H&E images.

An alternative aggregation method is building a graph on top of extracted features [47], [48]. This graph structure can be used as input to a graph neural network (GNN), which learns to make slide-level predictions. Methods using graphs are effective for whole slide images.

Another interesting technique is to extract many \( C \) features from every patch in an \( N \times M \) grid covering a WSI. By combining these vectors of features into an \( N \times M \times C \) ‘feature cube’, an aggregator CNN can be trained to label slides based on the feature cube for problems such as subtyping [49]. This was successfully applied by Tellez et al. [49] to predict the presence of metastasis and tumour proliferation speed at the slide level.

1.2.8 Challenges Facing the Advancement of CPath

Computational pathology is a rapidly developing field and still faces many challenges. One major difficulty when working with whole slide images (WSIs) is their large size, various file formats, and lack of standardization [19]. Infrastructure issues, such as the requirement of access to powerful GPU hardware clusters and high bandwidth interconnects for large model training, are also potential barriers for researchers [50]. Limited available memory for working with WSIs and training models necessitates working at a patch or visual field level, followed by aggregation [51] or using novel techniques
to incorporate wider context when making slide-level predictions [52]. Although some progress has been made to incorporate a larger context [49], [53] and enable learning with larger visual fields, this remains a challenge and new frameworks for incorporating a wider context are still needed.

The heterogeneity of some work is also an issue. While most ML workflows are Python-based, some libraries, such as BioFormats [54], are written in Java, which can necessitate using a language bridge. This can complicate environmental setup and add overhead when bridging between languages. Also, certain algorithm codebases, such as StarDist [55], utilize customized C++ extensions. C extensions in Python can be problematic due to platform-specificity, manual memory management, and difficulty in debugging.

In addition to the aforementioned challenges, licensing can also be an issue in ML workflows. Copyleft licenses like the GNU General Public License (GPL) may be avoided for fear of ‘infecting’ codebases [56] or limiting commercialisation [57]. This can limit the usage of libraries, such as BioFormats, in both research and commercial projects. Furthermore, there is no established common interface for various libraries, which can make integrating with multiple backends (for example, multiple whole slide image reading libraries) challenging. As a result, users often have to develop custom code to interact with different libraries, which can be time-consuming and error-prone.

Gathering training data from pathologists, who are experts in their domain, can be a major challenge as manual annotation of data
can be both costly and time-consuming. This leads to a shortage of high-quality labelled data available for training. Therefore, there is a growing interest in representation learning and self-supervised methods to make use of unlabelled data. Additionally, labelled data is known to have high inter- and intra-observer observer variability. Efficient web-based annotation tools are necessary to aid with data annotation and facilitate collaboration with pathologists. This is because NHS workstations can only access web applications via a browser and are typically restricted from installing additional software. However, with the large size of whole slide images, annotation sets can grow quite large, especially if they are generated with assistive annotation tools [58] or as the output from model inference. Efficient visualization of large annotation sets is challenging. Lastly, performing analysis of large sets of annotations can be time-consuming. For instance, a basic search for pairs of annotations that meet specific criteria, such as being located within a certain distance from each other, may be implemented using a pair of nested loops. However, this type of implementation has a time complexity of $O(n^2)$, which means that it becomes increasingly slow as the number of annotations, $n$, grows larger.

Understanding and explainability of algorithms remains a significant issue. The ‘black-box’ problem is a common challenge where the inner workings of some algorithms are not transparent, and it is difficult to understand how they reach their conclusions. This creates concerns about bias, fairness, and accountability. There are also many regulatory challenges when complying with regulations,
such as with the US Food and Drug Administration (FDA) and in the EU, that need to be addressed as computational pathology progresses. Quality control is also a significant challenge as clear guidance for regulatory compliance is needed. Furthermore, there is a ‘gold-standard paradox’ [59] in which algorithms are compared to human annotators as the ‘ground truth’, making it difficult to determine if superhuman performance is achieved. Furthermore, ethical and cyber security concerns such as data privacy, data ownership, and intellectual property are other major issues that need to be addressed as more healthcare data is generated and processed using AI algorithms.
1.3 Thesis Aims

The previous section outlines opportunities to enhance the effectiveness of computational histology. Therefore, the objective of this thesis is to develop tools for converting, analyzing, and annotating histology images. Some specific objectives include:

- **Facilitate rapid WSI conversion:** To provide user-friendly tools for fast and efficient conversion of WSI images. This may be for further analysis or compiling a homogeneously formatted dataset.

- **Improve accessibility to state-of-the-art CPath models:** by providing a suite of pre-trained models that can produce useful output in just a few lines of code.

- **Accelerate CPath pipeline research and development:** by creating a comprehensive collection of tools that address multiple stages of common CPath pipelines. This will streamline the development process and allow for faster iterations and improvements.

- **Assess the impact of compression on deep-learning models.** The research will explore how different compression techniques and compression levels affect the performance of deep learning models in histology image analysis. This assessment will provide insights into the optimal trade-off between image size reduction and model accuracy.

- **Investigate the potential for alternative representations**
of histology images: including unsupervised or self-supervised methods and other image transformations. This investigation aims to improve the generalizability of models and methods trained on histology images.

- **Provide tools for effecting annotation**: including space-efficient storage and rapid querying of large annotation data sets.

1.4 Thesis Organization

This thesis aims to contribute to the development of computational pathology (CPath) by addressing key challenges in converting, analysing, and annotating histology images. The following is a summary of the main contributions described in detail within the chapters of this thesis.

Chapter 2 describes **TIAToolbox, a computational pathology toolbox which provides an end-to-end API for histology image analysis**. The toolbox aims to make it easier for researchers to use, develop, and compare machine learning models for CPath. By providing pre-trained models and full replicable CPath pipelines, TIAToolbox enables pathologists and researchers to build upon existing work and accelerate the development of new CPath methods.

The toolbox provides a uniform API for reading many whole slide image formats, including emerging formats such as NGFF [60] and often poorly supported formats such as Omnyx JP2. It also provides several convenience methods, such as specifying the desired
output region size in physical units (microns-per-pixel, MPP) and synchronous reading of derived images, such as a tissue mask. Included implementations of pre-trained models such as HoVerNet [61] along with inference engines to handle the processing of WSIs and visual fields in just a few lines of code can jump-start the process of extracting diagnostic features from histology images for experimentation of downstream analysis. Other tools such as patch extraction, deep feature extraction, stain normalisation and graph construction methods provide a wide range of building blocks for building state-of-the-art CPath pipelines. In addition to these modules, extensive documentation and examples via Jupyter Notebooks demonstrate full implementations of published methods such as IDaRS [37] and SlideGraph+ [62]. Furthermore, the toolbox is extensively tested with over 99% unit test coverage and an extensive continuous integration system to ensure high code quality and reliability. As of writing, the toolbox has been downloaded over 130,000 times. This work described in Chapter 2 has also been published, peer-reviewed, and received positive feedback from the machine learning and CPath community.

Whole slide image conversion (WSIC) is a Python-based tool for converting WSIs showcased in Chapter 3. This tool aims to make WSI conversion easier and faster, especially for handling JP2 files which we found very slow to decode with open-source tools, including the reference implementation (OpenJPEG). WSIC also introduces a lossless repacking or transcoding between DICOM, TIFF, and NGFF (Zarr) formats. In addition to benchmarking the con-
version performance of WSIC, experiments on the impact that recompressing images during conversion has on deep learning model inference for the fine-detail task of nucleus instance segmentation and classification are explored. This identified that below a PSNR of 40 dB, there is a significant impact on model performance. Furthermore, JPEG compression could not achieve an image quality high enough to avoid significant variation from the results of inference on the original image. Overall, this work described in Chapter 3 addresses the issue of WSI conversion, which is an important pre-processing task for many CPath analyses and the organization of large CPath datasets.

In Chapter 4, self-supervised representations for CPath are explored using a swapped augmented view (SwAV) [63] method to train a CNN to produce image embeddings via clustering. Transfer of this learned representation to downstream tasks was demonstrated for tissue classification and cellular composition prediction. A proposed method for cellular composition regression (CellCoRe), trained on only a one-dimensional vector of cell type frequencies, provided accurate predictions approaching those of larger and more complex models trained on full pixel-level annotation data.

Further exploring alternative representations led to developing and using a pooled wavelet representation. This section on wavelet image representation used as input for a modified residual network (ResNet) architecture explores the potential of using the discrete wavelet transform (DWT) and max pooling to represent large tissue regions. This approach works well for colorectal ade-
nocarcinoma grading and lung cancer subtyping and could be used for other tasks. Furthermore, it is evaluated against other methods which allow for large visual field sizes, such as bilinear downsampling and a block-wise discrete cosine transform.

In summary, Chapter 4 explores and demonstrates a self-supervised learnt representation for downstream patch-wise tasks and an alternative representation for large visual fields, which could avoid the need for patch aggregation or patch-level annotation.

Chapter 5 describes tools developed for creating, visualising, storing, and querying annotations for histology images. In particular, the implementation of **space efficient storage and fast query methods**. This chapter on WSI annotation proposes an efficient method for storing and querying annotation data via SQLite and a spatial R*-tree index. This chapter also investigates and evaluates methods of serializing annotation geometry and ways to improve query speed for common use cases such as neighbourhood-based analyses, such as tumour-infiltrating lymphocyte (TIL) detection. The chapter also introduces a simple Domain Specific Language (DSL) to abstract the query from the backend, which can allow for query portability and future extension to new backend implementations.

Finally, Chapter 6 summarizes all the main findings and contributions of the thesis and discusses their implications. It reflects on the impact of the research by analyzing factors such as downloads, citations, and social media engagement. The chapter also provides insights into potential future work that can build upon the current
research. By considering the limitations and scope, the conclusions chapter offers suggestions for further research directions and areas where improvements can be made.
Chapter 2

TIAToolbox as an End-to-End Library for Advanced Tissue Image Analytics

2.1 Introduction

Digitization of classical cellular pathology workflows through the deployment of digital whole slide image (WSI) scanners has resulted in significant progress in the development of computational pathology (CPath) image analysis techniques. Such advances have benefited greatly by adapting deep learning techniques from computer vision producing novel solutions to a variety of CPath problems, including nucleus instance segmentation [61], pathology image quality analysis [31] and WSI-level prediction [37], [47]. Although numer-
ous algorithms have been developed for the analysis of WSIs, many of which share common components such as WSI reading, patch extraction, and feeding to deep neural networks, there is no single open-source generic library that unifies all the steps using best practices to process these images. Several published algorithms have their own packaged codebases which run in a task-specific environment, with tightly coupled interfaces, dependencies, and image format requirements. It is also common for there to be little to no code quality checks or unit testing. This may prevent code from a published peer-reviewed method from being able to run out of the box, decrease the reproducibility of experiments, handicap the ability to extend or adapt existing methods and increase the time required to understand the codebase. TIAToolbox is a suite of unit-tested image analysis and machine learning (ML) tools developed for the CPath community, making it possible for a variety of users to construct and reproduce CPath analytical pipelines with cutting-edge methods.

The main objective of this work was to provide an open-source library to the CPath community, which is simplified, streamlined, reproducible, easy to use, and unit-tested. We aim to allow researchers to build their analytical pipelines on state-of-the-art methods. To achieve this, we provide a simple to use Application Programming Interface (API) which abstracts unnecessary complexity from the user where possible. This means that the API users can write code with a focus on the task at hand instead of being distracted by unnecessary details or peripheral tasks, such as managing multiple
processes or needing to know the details of different WSI formats. The WSI reading capability of the toolbox is a good example of such an abstraction that simplifies WSI reading. It hides unnecessary details of various file formats while keeping intact important format-related metadata required for ML tasks. For the reproducibility of algorithms, we provide pre-trained published benchmark algorithms which can be run using only a few lines of code. This can help researchers to build on state-of-the-art methods and greatly simplifies the reproduction of previous results. Weights for these pre-trained models can be automatically downloaded at runtime or can be provided by the user, making it easier to test alternate models using the same pipeline. We posit that TIAToolbox will help establish objective and measurable standards of progress in the development of CPath algorithms.

One of our main guiding principles is to make CPath accessible to researchers without expertise in deep learning for CPath-specific tasks. We provide example notebooks (https://github.com/TissueImageAnalytics/tiatoolbox/tree/publication/examples) for this purpose. These notebooks can be run in a web browser on local machines or free-to-use platforms such as Google CoLab and Kaggle. The online platforms require no local installation and are well-suited to non-technical users. The notebooks additionally serve as a manual by example for the use of the TIAToolbox. Our toolbox is supported by extensive online documentation (https://tia-toolbox.readthedocs.io/en/publication), including examples, for each module in TIAToolbox. In addition, we provide
a command-line interface that enables experienced programmers to use the components of the package in Bash scripts and to batch-process their images or WSIs on CPU/GPU clusters.

In this section, we provide a brief review of existing tools for reading whole slide images (WSIs), image annotations and image analysis. Image reading libraries, such as OpenSlide [64] and Bio-Formats [54], allow the reading of WSI image formats. However, OpenSlide does not support several image formats. For example, it is unable to read JPEG-2000 JP2 images (although it can read JPEG-2000 J2K TIFF tiles) generated by legacy GE Omnyx scanners and images in OME-TIFF format (https://docs.openmicroscopy.org/ome-model/5.6.3/ome-tiff/), a commonly used open and well-documented file format. BioFormats supports the reading of many WSI image formats. However, it is a Java library making it potentially difficult to integrate with Python-based workflows. The Java Python interface of BioFormats allows one to bridge this gap. However, it can be slow, complicated to set up and requires a variable set of parameters for different WSI formats – not ideal for a newcomer. Additionally, when reading JP2 images BioFormats relies on an outdated and unmaintained implementation from the Java Advanced Imaging (JAI) library for which support and documentation from Oracle have been discontinued. QuPath [65] provides a graphical user interface and the ability to read a variety of formats. However, because of its dependence on Java, its integration with a custom Python ML pipeline may require additional steps.

Although it is possible to use separate libraries for various for-
mats, different interfaces and resulting data types can make writing code to handle multiple formats complex and error-prone; especially when trying to replicate existing algorithms. This causes a significant loss of researchers’ time in handling technical issues instead of evaluating and developing new pipelines. There are other considerations, such as handling metadata from various formats, re-sampling of images, integration with image processing tools and optimizing data loading from machine learning libraries.

QuPath includes some classical image processing algorithms and also integrates with some DL models as plugins. For example, it includes a semantic pixel segmentation method which utilizes a user-configurable set of simple image features (e.g., colour channel intensity, gradient magnitude, Laplacian of Gaussian, etc.) which are fed to specified classifiers such as a random forest, k nearest neighbours (KNN), or artificial neural network (ANN). Pre-trained DL models, for example, StarDist [55], are not included directly with QuPath but may be downloaded by a user and enabled as a plugin.

DL models typically produce results of higher quality than classical image processing methods, due to their ability to automatically extract representative image features. Therefore, we focus on including pre-trained cutting-edge pre-trained models in TIAToolbox which have been trained on images sampled across many slides using large public data sets, making them easily usable without any further user configuration or labelling.

Other Python software packages, such as PathML [66], offer some trained deep-learning models. However, the selection is often lim-
ited, currently only one model (HoVer-Net) in the case of Dana-Farber-AIOS PathML, with a U-Net [40] implementation in progress. There is also no clearly documented way to integrate additional models or custom user models with PathML.

It is common for histology image analysis packages (such as HEAL [67], HistoCartography [68] and CLAM [38]) to focus on a particular method, model or approach. In contrast, TIAToolbox can integrate with standard PyTorch modules (including many third-party PyTorch-based modules) and does not require the use of custom TIAToolbox layers or modules within the model architecture definition. It allows batch processing of several hundreds or thousands of WSIs and employs a modular structure, allowing for a wide variety of techniques to be integrated with the toolbox and for its modules to be used as components in new analytical pipelines.

TIAToolbox addresses the aforementioned issues and provides a broad feature set, shown in comparison with other histology-focused software packages. The main contributions of TIAToolbox are as follows: development of histopathology image analysis pipelines, support for a wide range of WSI formats, a unified framework, efficient image reads, tile generation (Zoomify), modularity, high unit-test coverage (> 99%), reproducibility of state-of-the-art methods, cross-platform compatibility (Windows, Linux, and macOS), ease of use, a command line interface (CLI) and a pure Python/CPython source. Our toolbox provides the most extensive integrated solution to a variety of important histopathology image analysis tasks ranging from multi-format image reading, patch and tile extraction, stain
normalization, instance segmentation, patch classification and extraction of deep features for the development of WSI-level weakly supervised prediction models through weakly-supervised and graph neural network techniques as well as visualization of their results.

It has the functionality to read common WSI image formats including OpenSlide compatible WSI formats (including Aperio SVS, Leica SCN), OME-TIFF (OMERO) and JP2 (Omnyx) in addition to visual fields (JPEG, PNG) using a single Python API in a unified framework. Furthermore, it also allows the addition of other existing and newly emerging formats.

Random-access reading and re-scaling of these WSIs based on resolution metadata (e.g., microns per pixel) are done efficiently, making use of multiple stored resolutions. This allows efficient implementation of multiple instance learning (MIL) algorithms such as IDaRS3 that require random sampling of tiles. Designing the toolbox to be composed of modular reusable components encourages the development of new analytical pipelines. We integrate and verify published models using these modules in addition to providing pre-trained weights to enable the reproduction of results.

We use abstraction where possible to reduce complexity for new users and to enable users with little programming experience to perform common tasks (such as shown in Appendix A.1) without having to worry about awkward edge cases. When implementing tools or integrating existing tools, we test for compatibility across Windows, Linux, and macOS. In addition, we also provide many web-based example notebooks to run the code.
Lastly, there is no need to bridge between languages, such as between Java and Python. Only Python code or CPython compatible C/C++ extensions are used. Language bridges can be problematic to set up and often have performance issues. Therefore, we have avoided requiring one for the toolbox to function.

In summary, we present an open-source unit-tested, unified cross-platform software library with comprehensive tools for WSI reading, patch extraction, pre-processing, model inference, post-processing, and visualization. We provide a platform for reproducible computational pathology using classical machine learning and advanced deep learning for end-to-end tissue image analysis.

2.2 Methods

2.2.1 Reading WSI Data

TIAToolbox provides a common interface for random-access reads of image regions from disk using an API defined in an abstract base class. Readers providing support for specific formats are implemented by sub-classing the base reader. We currently support reading a variety of tagged image file format (TIFF) based WSI images (including SVS, SCN, NDPI, MRXS and generic tiled TIFFs) using an OpenSlide [64] backend, OME-TIFF files using a tifffile (https://www.lfd.uci.edu/~gohlke/) backend and reading from JPEG 2000 based slide formats (such as JP2 files generated by GE Omnyx scanners) using the Glymur (https://github.com/quintusdias/glymur) and OpenJPEG (https://www.openjpeg.org) as a back-
end. We also provide preliminary support for the rapidly evolving Zarr format (https://zarr.readthedocs.io/en/stable). Lastly, we include support for reading WSI DICOM images (via wdidicom) with JPEG and JPEG2000 compressed tiles. Furthermore, we include experimental support for a developing next-generation file format (NGFF version 0.4) based on Zarr [60].

The reader class implements read functions based on physical resolution units, such as MPP or apparent magnification. This is useful to reproduce the results of published algorithms which might have been trained at a specific magnification or MPP. For example, a read can be performed with the resulting image scaled to 0.5 MPP or an apparent magnification of $20 \times$. For efficient image reads, we use pre-computed lower resolution copies when reading to avoid costly and unnecessary re-sampling of large image regions when re-scaling to the user-requested resolution and units. This is done using metadata specifying the physical resolution of the WSI and down-sampled copies of the image embedded in the WSI file. The standard image pyramid is illustrated in (Figure 1) 2.2.1, which depicts multiple copies of an image stacked on top of each other in decreasing resolution.

When reading a region from the WSI, we define two modes of operation: the read_bounds mode that allows reading with a fixed field of view as resolution varies and the read_rect mode with a fixed output size as resolution varies. The read_rect method accepts a location and output size as arguments. This method is useful for situations where the output must remain the same size, for ex-
Figure 2.2.1: Different resolutions, stored in the WSI, are shown as blue planes stacked on top of each other. A lower resolution is a stored down-sampled copy of the highest resolution (baseline). Here both read modes, read_rect and read_bounds, illustrate reading a region of interest containing some tissue (magenta shape) at a desired resolution. Reading of a region which is not at a pre-computed and stored resolution within the WSI (transparent white plane with a dashed outline) results in a read via a down-sample interpolation from a level with higher resolution.

ample, while extracting patches. As illustrated in Figure 2.2.1, this results in a changing field of view as resolution varies. Conversely, read_bounds ensures a fixed field of view at all resolutions but may result in a different output image size. This is useful if there is some tissue feature which must be isolated in the view. To the best of our knowledge, no other tool provides equal flexibility in manipulating WSI pixel data.

Our advanced WSI reading tool easily fits within various CPath pipelines due to the wide range of image formats that it supports. This is demonstrated in the patch aggregator and graph aggregator pipelines as presented earlier, where the same reading functionality is incorporated. To help researchers easily use our toolbox for WSI
reading, we provide a specific notebook (see Example Notebook 01) with multiple examples.

2.2.2 Virtual Whole Slide Image Pyramid

A virtual WSI reader class enables reading image data from single-resolution visual fields, such as JPEG or PNG files, using the same interface as defined for reading WSIs. This facilitates the creation of a virtual image pyramid similar to the WSI pyramid in Figure 2.2.1. An effective use case for this is when reading from an image derived from a WSI, such as a tissue mask or patch classification output map. These images are typically at a much lower resolution than the full-size WSI. A virtual image pyramid can have pyramid levels specified for which there is no stored re-sampled image, or which have larger dimensions than the image data itself. However, when read using the WSIReader interface, the virtual WSI will behave as if those resolution levels do exist simply by interpolating the available image data. As a result of this behaviour, the original tissue WSI and a derived image can then be read synchronously, using the same coordinates and resolution arguments as shown in Appendix A.2, simply by copying the metadata about available resolutions and the physical scale (MPP) of the baseline resolution. This relieves the user of having to perform cumbersome and error-prone conversions between different coordinate systems.
2.2.3 Metadata

Metadata format varies greatly between file formats. We cater for this when initializing the reader object by creating a metadata object and thus providing a unified object when accessing image file related metadata. Since this is implemented as a Python class, static analysis tools common in many integrated development environments can parse it and offer helpful auto-completion suggestions, making it easier for researchers to write and implement their pipelines. The original underlying metadata is stored so that it remains accessible if required. Additionally, important metadata such as MPP may be specified if it is not found within the file. This is commonly useful when reading large visual fields or creating virtual WSIs which have a known magnification or MPP but are missing embedded metadata.

2.2.4 Tissue Masking

Most WSIs contain a large amount of background area (e.g., glass, slide vendor name etc.) which is of no biological significance and can be ignored to speed up downstream processing and analysis. To identify these areas a tissue mask is commonly generated. In fact, tissue mask generation is common practice and is used in most CPath applications, including those presented in the Results section. We include some basic methods for creating such tissue masks based on Otsu thresholding [30], which separates pixels into foreground and background by minimizing the intra-class intensity vari-
ance. We show how one can combine Otsu’s method with some basic morphological operations to remove small holes and regions. These masking classes can easily be extended to more advanced methods by creating a subclass of the abstract base class. A convenient function is provided to quickly generate a virtual WSI of a mask from a tissue WSI at a desired resolution. A notebook on tissue mask generation can be found within the TIAToolbox repository (see Example Notebook 03). We also provide a DL-based method for tissue masking, which is described in more detail later in the Semantic Segmentation section.

2.2.5 Patch Extraction

It is common to apply DL models using small images [61], [69] due to GPU memory constraints and required model complexity. As such, it is also common to divide a large WSI into small patches for training and inference with a model. This could be done simply by iterating over the WSI dimensions with a stride equal to the desired output patch size required by the model. However, there are several additional things to consider. Firstly, since pathology images are calibrated and have a known scale, patches may be extracted at a specific resolution (for example 0.5 microns-per-pixel). Our patch extractor rescales to the desired output resolution. Additionally, it can handle edge cases, such as whether to include patches which would partially extend beyond the edge of the WSI. Our patch extraction module can flexibly handle such edge cases by either discarding these patches or padding them to maintain a homogeneous
output size. Also, an overlap can be specified so that each extracted patch partially overlaps its neighbours. The patch extractor, implemented as an iterator, can extract patches as needed which avoids filling available memory with patches until they are needed resulting in increased memory efficiency. In addition to grid-based patch extraction, patches may be extracted around each point in a set of coordinates. This is particularly useful for extracting patches centered on known cell nucleus locations or randomly distributed patches across the WSI. The PatchExtractor also supports functionality to filter out non-tissue regions while generating patches. To highlight the effectiveness of our efficient patch extraction tool, we provide an easy-to-follow interactive notebook with multiple examples (see Example Notebook 04).

### 2.3 Stain Normalization & Augmentation

It is well known that digital pathology images vary in colour appearance due to factors such as differences in scanner manufacture and variation in tissue preparation. For example, thicker specimens tend to stain the tissue darker. Differences in temperature, stain concentration, duration of staining and scanner type and settings can also lead to stain variation. This may harm the performance of automated methods unless dealt with appropriately.

It is possible to perform simple colour normalization using first-order statistical measurements but doing so may not correctly model the variation in stain appearance. A commonly used pathology-
specific pre-processing step is to perform the separation of histo-
logical stains into separate optical density (OD) channels from the
original red, green, and blue (RGB) sensor data and optionally ap-
ply normalization across the OD channels. TIAToolbox includes
several commonly used methods for normalization, including Rein-
hard [70], Macenko [14] and a modified Vahadane [15]. The toolbox
implementation is adapted from the StainTools [71] Python pack-
age created by Byfield. Our implementation of Vahadane’s method
exchanges the SPArse Modelling Software (SPAMS) [72] dictionary
learning step with an equivalent implementation in scikit-learn [73]
and SPAMS LARS-LASSO regression with ordinary least squares
(OLS) regression. We do this to maintain cross-platform compati-
bility and for speed of execution. Other implementations of LARS-
LASSO, for example in scikit-learn, performed orders of magnitude
more slowly. We demonstrate how a user can use stain normaliza-
tion in their pipelines by providing a descriptive Example Notebook
(02).

Instead of normalizing image data, another method used in com-
putational pathology is stain augmentation. This is particularly
useful when training DL models to increase a model’s robustness to
stain variation. In TIAToolbox, we leverage stain extraction meth-
ods described above to randomly perturb the Haematoxylin and
Eosin stain contents of each image used for training purposes. We
also ensure the integration of our stain augmentation functionality
into commonly used augmentation packages, such as albumentations
[74].
2.4 Models

Each CPath pipeline usually contains numerous steps and requires special consideration so that large-scale WSIs can be effectively dealt with. In fact, recent state-of-the-art models in computer vision for tasks such as segmentation [69] and classification [75], [76] cannot be directly used when working with multi-gigapixel inputs due to memory limitations. This is due to the lack of available tools that can handle WSIs effectively in machine learning pipelines because of their large spatial dimensions. As the WSIs commonly get divided into smaller independent image patches, each processed by a machine-learning model before merging the patch-level results, it is common practice to build custom tools from the bottom-up (i.e., starting from patches) to tackle such challenges.

Despite an increase in the number of models provided within CPath, model weights are not always available. Even when weights are provided, downloading and management can become challenging when working with multiple code repositories. Current DL libraries [77], [78] enable seamless downloading of models, along with their parameters, yet these models have not been trained on problems within CPath. Even if these models were trained on task-specific data, additional work would still be required for use with WSIs.

To help overcome the above shortcomings, we provide an easy-to-use API where researchers can use, adapt and create models for CPath. TIAToolbox enables researchers with different levels of experience to easily integrate advanced CPath algorithms into their
research projects. Once again, this avoids having to reinvent the wheel. We aim to achieve these goals by: Introducing a common API to assemble predictions for common CPath tasks, such as: instance segmentation, semantic segmentation and classification; integrating several well-established models (pre-trained weights and model code) for the above tasks; utilizing a common data loader to seamlessly load WSIs within each model irrespective of the task at hand.

2.4.1 API for Models

To enable the integration of multiple models within the toolbox, we implement a common API, comprised of three components: a Dataset Loader, Network Architecture and Engine. The Dataset Loader defines how the data is sampled and converted into batches. The Network Architecture contains the model architecture, defines how to process an input batch and specifies how to post-process the results. An Engine defines how the Network Architecture and Dataset Loader interact, runs inference and assembles the output into a WSI-level prediction.

In the above three components, the Dataset Loader and Engine are designed in such a way that they should not need to be modified unless performing a task not supported by TIAToolbox. In our initial release, supported tasks include patch classification (PatchPredictor), semantic segmentation (SemanticSegmentor) and nucleus instance segmentation and classification (NucleusInstanceSegmentor). As described above, the Network Architecture defines the in-
Figure 2.4.1: The framework comprises three main components: dataset loader, network architectures and engine.

Interaction of various network layers and determines how to transform the output into the final prediction via post-processing. We typically include post-processing within the network definition because this can often be model-specific. For example, nuclear instance segmentation models may produce different outputs and therefore need to be processed according to the type of output generated. We demonstrate how the Dataset Loader, Network Architecture and Engine interact in Figure 2.4.1. Here, we observe that the Dataset Loader and Network Architecture are provided to the model Engine, where the data is then processed in the backend by the inference and aggregation flows.

In our toolbox, we support a handful of different models, such as ResNet [75] and DenseNet [76] for patch classification, U-Net [40]
for semantic segmentation and HoVer-Net [61] for nuclear instance segmentation and classification. We also provide an extension of HoVer-Net that performs segmentation of additional regions using a single network [79]. We have designed the API in such a way that using a custom model in place of our supported models is straightforward. Therefore, researchers can focus solely on model development because the handling of WSI data is done behind the scenes by the Dataset Loader and Engine. With just a few lines of code, supported models can be used without modification. As part of our toolbox, we provide detailed examples that describe how to easily use both pre-defined and custom-built models for a given application.

Below we provide more information on the three main tasks initially supported in the toolbox: patch classification, semantic segmentation and nuclear instance segmentation and classification. The three tasks are similar in that they make a prediction for small image patches before aggregating the results. However, they differ in the type of output that is produced. For all these tasks, we provide detailed interactive example notebooks that clearly describe how to implement each of the models described in this work. Sample outputs obtained using TIAToolbox for both semantic segmentation and nuclear instance segmentation and classification can be seen in (Figure 2.4.2).

2.4.2 Patch Classification

Due to the sheer size of WSIs, DL methods in CPath often involve making a prediction based on smaller image patches. To as-
Figure 2.4.2: a) H&E stained input visual field. b) Semantic segmentation output. c) Nucleus instance segmentation and classification output.
sist with this, we provide a framework for patch-based classification, which can process image patches, larger image tiles or WSIs as input. Working with these different input types is streamlined in our toolbox and simply requires a user to define the input type as an argument in the code, as shown in Appendix A.3 and Appendix A.4. When the input is an image tile or WSI, the model will process each patch within the model consecutively and then aggregate the results to give a result for each patch within the input. The default post-processing scheme makes a patch-level prediction by selecting the class with the highest probability. The final output returns the path to a file that specifies the model predictions and the corresponding patch coordinates within each WSI image. When passing WSIs as input to our patch predictor, the toolbox internally uses the PatchExtractor class to obtain patches for the prediction model. Arguments for this extraction are passed through from the predictor initialization to the extractor.

When using models trained to predict the tissue type in colon tissue, the model will predict an input image patch to be one of the following 9 classes: background, normal mucosa, tumour, inflammatory, debris, muscle, mucous, stroma or complex stroma. A full list of the available DL models for patch classification is given in Appendix B.1. For the breast tumour classification dataset, we used the PCAM training and validation splits. However, for the colorectal cancer dataset, we created our own data split to speed up the inference time. We show the validation results obtained after training each model on the two patch classification datasets in
Supplementary Table 2. We also highlight the ease of use of our patch predictor by integrating it within our example pipeline on the prediction of key mutations and molecular pathways.

2.4.3 Semantic Segmentation

It is often desirable to localize regions within an image, rather than assigning a value to an entire input patch. This enables a more precise delineation of region boundaries and allows morphological features to be extracted from the tissue. Semantic segmentation localizes regions, without separating touching objects belonging to the same class. This may be sufficient when analysing different tissue regions, such as tumour and stroma and the aim is not to extract subsequent features from individual objects, such as glands and nuclei. As in the case of the patch classification model, our semantic segmentation framework processes input patches separately, before merging the results. The difference here is that a prediction is made per pixel, rather than for the entire image patch. Despite this, the API remains similar between the patch prediction and semantic segmentation tools, as can be seen in Appendix A.5. The output of the model is a 2-dimensional map of the segmentation prediction, at a resolution specified by the user.

In the toolbox, we provide a U-Net based model with a ResNet50 backbone, trained on a multi-class breast cancer semantic segmentation (BCSS) dataset[80]. Here, the model will predict pixels to be one of tumour, stroma, inflammatory, necrosis, or other. In Supplementary Table 3, we report the Dice score for each class obtained
by our model after training. We compare these scores to those obtained in the original paper and observe that overall, we achieve a better performance in terms of average dice score over all classes. Note that despite the models not being identical, they both use a U-Net architecture with a ResNet50 backbone. In addition, we train the same model architecture for the task of tissue masking, which can enable a more precise result than conventional threshold-based methods (see Example Notebook 06).

2.4.4 Nuclear Instance Segmentation and Classification

Identification and localization of different nuclei is a particularly important task in the field of CPath because it enables subsequent extraction of cell-based features that can be used in various downstream tasks, such as cancer grading [81] and biomarker discovery [82], [83]. Identification and localization of different nuclei is a particularly important task in the field of CPath because it enables subsequent extraction of cell-based features that can be used in various downstream tasks, such as cancer grading [81] and biomarker discovery [82], [83]. Here, it is necessary to separate clustered nuclei at the output of the model to ensure that features inferred from the model output correspond to individual nuclei. Classifying the types of nuclei can help profile the tumour microenvironment because it enables the quantification of different types of cells in various areas of the tissue. For this task, like other tasks defined in TIAToolbox, individual patches are processed before merging the results. However, a more complex post-processing step is needed to ensure
individual nuclei are effectively separated and classified into distinct categories.

For this task, we provide a top-performing approach for nuclear instance segmentation and classification within TIAToolbox, developed by members of our research group. The model, named HoVer-Net, has been increasingly used in recent research projects [47], [68] in CPath, due to its state-of-the-art performance across a range of different datasets. In the toolbox, we include nuclear instance segmentation models trained on the PanNuke [84], [85], CoNSeP [61] and MoNuSAC [86] datasets – three widely used datasets for instance segmentation and classification of nuclei. For this, we use the original model weights and therefore, we encourage readers to refer to the original papers for details on performance. Further information on the predicted classes when using models trained on the aforementioned datasets is provided in Supplementary Tables 1-8. We demonstrate how easily our nuclear instance segmentation tool can be integrated into CPath pipelines by demonstrating how it can be seamlessly used during our graph aggregator example. Again, our nuclear instance segmentation and classification tool is simple to use and uses an API in line with other models in the toolbox. This can be seen in Appendix A.6.

In addition, we provide HoVer-Net+ [79], which extends the original HoVer-Net model by adding a fourth decoder to perform the task of region-level semantic segmentation. In particular, the model that we provide in the toolbox has been trained on a private cohort of oral epithelial dysplasia WSIs to segment various nuclei (e.g., ep-
ithelial, inflammatory) and the different intra-epithelial layers. For further information on performance, we ask readers to refer to the original publication [79].

2.4.5 Customizing Models

In the toolbox, we supply model architectures along with associated pre-trained weights to enable models to be used out-of-the-box. However, it may be desirable to use one of our defined model architectures, but with different weights. For example, users may train a model on a different dataset, or a different training strategy may be used to obtain the weights. If a user would like to do this, the default pre-trained weights may be overridden by simply adding the path to new weights as a class initialization argument. We show an example of how this can be done in Appendix A.7. Furthermore, TIAToolbox is flexible and is designed to allow users to add their own PyTorch compatible models for any of the tasks included within the toolbox. We provide sufficient examples in the form of interactive notebooks (See Example Notebook 07) to detail the steps required for model customization.

2.4.6 Deep Feature Extraction

In many CPath pipelines, it is of interest to extract deep features from input images, which can be used for downstream tasks, such as clustering [87], patch classification [38] and graph-based learning [62], [68]. Visualizing deep features can also help us to better understand which areas within an image the model may be focusing on,
which can help further guide researchers with model development. Deep features are obtained by passing an image through a trained CNN and extracting the features immediately before the classification layers. A popular strategy is to utilize networks trained on the ImageNet dataset because they are optimized on millions of example images and thus are likely capable of extracting strong features. Therefore, we ensure that ImageNet-trained models can be integrated with TIAToolbox, enabling the extraction of strong deep features for downstream tasks. In future, we plan to support extracting deep features using additional datasets and different optimization techniques such as self-supervised learning.

2.4.7 Visualization

We provide several convenient functions for visualizing the results of model predictions. These include merging of prediction outputs and overlaying predictions on the predictor input image (Example Notebook 05) and plotting a generated graph (see Appendix A.8). Our toolbox also implements generating multi-resolution tiles in a format commonly used by interactive web-based (slippery map) viewers such as OpenLayers (https://openlayers.org) where a tile server streams image regions on-demand to a web client, for the display of very large images and geospatial data which can be panned and zoomed by a user. We additionally include a simple web application that can be viewed in a web browser. An example of this is shown in Appendix A.9. This can also be used in combination with the functionality of a virtual WSI to allow for ease of visualization, such
as overlaying patch predictions on top of a WSI.

2.5 Ethical Approvals for Datasets

We built our software on datasets which are previously published or publicly available. Therefore, no additional ethical approval was required for this work.

2.6 Results

In this section, we demonstrate the utility of TIAToolbox for two WSI-level prediction tasks, using recently proposed DL models, while demonstrating several of the other functionalities of the toolbox. First, we predict the status of molecular pathways and key mutations in colorectal cancer from Haematoxylin and Eosin-stained (H&E) histology images using a two-stage patch-level classification model. Next, we predict the HER2 and ER status from H&E histology images using SlideGraph+, a graph neural network-based model. We show that the implementation of both pipelines has been simplified using a common interface provided by TIAToolbox as shown in Figure 2.6.1. This reduces the effort needed by a new researcher seeking to extend these approaches.
Figure 2.6.1: The diagram shows the main steps of the SlideGraph and IDaRS pipelines and how modules in TIAToolbox have been used to replicate these pipelines. 

a) Simplified block diagram of the main steps involved in each of the example pipelines. Several of the steps at the start of these pipelines are common between the two methods and are provided by TIAToolbox. Additionally, many of the steps where the pipelines diverge are also included in the toolbox. Custom code is only required for one or two steps in each pipeline in addition to gluing together each of the pipeline stages, or for some custom visualization. The same pre-trained models can be used for inference in both IDaRS and SlideGraph+ pipelines.

b) The main steps of the IDaRS pipeline for an example WSI. For each input patch a mutation prediction (positive or negative) is made and the results are merged. Each component of the output vector is represented as a plane in a stack.

c) An example WSI and the resulting graph from the SlideGraph+ pipeline. Nodes are coloured in RGB space via a uniform manifold approximation and projection (UMAP) of the feature vectors assigned to the nodes.
2.6.1 Patch Aggregator: Predicting the Status of Molecular Pathways and Mutations using Patch-level Predictions

Assessment of the status of molecular pathways and key genetic mutations helps better understand the patient prognosis and can provide important cues for treatment planning. Typically, this assessment is done via genetic (e.g., polymerase chain reaction or PCR) or immunohistochemistry (IHC) testing. However, these tests may lead to time delays and additional costs because they are used as an extra step after initial analysis on routine H&E-stained slides. Recently, it has been shown that deep learning has the potential to predict the status of pathways and mutations directly from the H&E slides, potentially bypassing the need for additional tests [21], [37].

Despite the obvious advantages of H&E based prediction using deep learning, some researchers may struggle to reproduce the mutation prediction pipeline, where slight changes in the code may lead to vastly different results. Furthermore, new researchers may be discouraged from implementing the method, due to the challenge of working with high-dimensional histology data. Here, we show that TIAToolbox can be used to complete all necessary steps to predict the status of molecular pathways and key mutations in colorectal cancer and help simplify the overall analytical workflow. To achieve this, we follow the same approach used in the original paper by Bilal et al. [37] and use a two-stage pipeline. We first localize the
tumour regions to identify the potentially diagnostic areas and then use the IDaRS model of Bilal et al. [37] to make a prediction for the entire whole-slide image. Using the toolbox, these two steps can be completed with reproducible results without the need for advanced programming experience. We display our entire simplified IDaRS integration into TIAToolbox in Appendix A.11. It is worth noting that both stages use the toolbox’s PatchPredictor, as shown in Supplementary Note 4 and differ only in terms of the pre-trained model, which is defined during class initialization.

Identifying the tumour regions as an initial step is important for various tasks, for instance enabling the downstream analysis to be focused on diagnostically relevant areas. This initial step may also be useful in other tasks, such as cancer staging [88] and cancer subtyping [89]. To help overcome challenges resulting from limited computer memory, it is common to divide multi-gigapixel WSIs into smaller image patches, which are processed independently before merging the results. Using this approach, we obtain a tumour detection map by determining whether each input patch within the tissue contains any tumour. We utilize a pre-trained ResNet [75] within TIAToolbox’s PatchPredictor model to efficiently deal with patch-level processing and aggregation.

After obtaining the tumour detection map, we follow a similar divide, process and merge approach to obtain the task-specific prediction map. Using TIAToolbox’s patch prediction functionality, each tumour patch is seamlessly processed with a pre-trained ResNet and the results are merged. This prediction map can help improve the
interpretability of results made by IDaRS and identify areas contributing to the overall prediction. To obtain the final WSI prediction, patch results are aggregated to give a single score. IDaRS is a weakly-supervised approach, trained using a multi-instance learning technique and therefore the slide-level score is obtained using a common pooling strategy, such as selecting the maximum or average probability over all tumour patches. In the toolbox, we provide models trained on the first fold used in the original paper by Bilal et al. [37] to predict the following: microsatellite instability, hypermutation density, chromosomal instability, CpG island methylator phenotype (CIMP)-high prediction, BRAF mutation and TP53 mutation.

As a result of TIIAToolbox taking care of complex WSI handling behind the scenes, this pipeline has been reproduced in Example Notebook (IDaRS), utilising the same code fragment as in Supplementary Note 4 with the toolbox as the backend uses significantly fewer lines of code than the original implementation. This highlights how functionalities in the proposed toolbox can be efficiently leveraged for WSI prediction tasks in CPath. These patch prediction models can use individual patches, larger image tiles or WSIs as input. For this example and to follow the approach used by Bilal et al. [37], we choose to focus on WSI-level inputs. To help reduce the inference time, the models that we include within the toolbox have been retrained without stain normalization, as opposed to the original IDaRS implementation. A full breakdown of performance obtained after retraining is provided in Supplementary Table 4. We
observe that despite a slight reduction in performance which may be due to not using stain normalization, models provided with the toolbox can successfully predict molecular pathways and mutations.

2.6.2 Graph Aggregator: Predicting HER2 Status using SlideGraph+

HER2 and ER status are key prognostic indicators for establishing an appropriate breast cancer treatment plan. As with other biomarkers, they are typically assessed with IHC staining. Instead, determining status via routine H&E slides can potentially reduce costs and time to treatment. We show the integration of SlideGraph+ [62] pipeline using TIAToolbox for the prediction of HER2 status and ER status from H&E-stained histopathology images. SlideGraph+ is a message-passing graph neural network-based pipeline for WSI-level prediction that works by modelling each WSI as a graph with nodes corresponding to tissue regions and each node having a set of local features. Edges between nodes represent spatial organization within the tissue (see Figure 2.6.1).

The SlideGraph+ pipeline consists of patch extraction from WSI(s), stain normalization, node-level feature extraction, graph construction and prediction of the WSI label via a graph convolutional network (GCN). It is perhaps worth noting that this graph-based method is generic and can be applied to a wide range of WSI classification problems, as it is agnostic to both the problem at hand and the features utilized for prediction.

As shown in Figure 2.6.2 and Figure 2.6.1, the IDaRS and the
SlideGraph+ pipelines have numerous modules in common. Many of the same modules used in the IDaRS pipeline can be reused without reimplementation of the whole pipeline. Using PatchExtractor and StainNormalizer, it is easy to extract patches from tissue regions of the WSI and apply stain normalization across patches in the same way that is done in the original SlideGraph+ implementation. For each of these patches, a set of features must then be extracted. Different types of features can be extracted here, such as deep features from a CNN pretrained on the ImageNet dataset, cellular morphological features (class, major axis diameter, eccentricity, etc.) derived from the HoVer-Net segmentation and classification output, or the output of a network trained to regress from an H&E patch to the corresponding DAB intensity between registered H&E and IHC slides, as demonstrated by Lu et al. In our example implementation, we use TIAToolbox’s DeepFeatureExtractor to obtain features from an ImageNet pre-trained ResNet and cellular morphology features from HoVer-Net. Here, the state-of-the-art HoVer-Net model is provided as part of the toolbox’s NucleusInstanceSegmentor engine, which can be used to subsequently obtain either deep or cellular composition features. The modularity of the toolbox and the flexibility of the SlideGraph+ method allows for fast and easy experimentation, without having to write a lot of code to reimplement many common steps like patch extraction, stain normalization and feature extraction.

TIAToolbox provides a hybrid clustering graph construction method, as used by Lu et al. [62], which requires only the location of each
patch within the WSI and a corresponding node-level feature vector. This method clusters patches based on a weighted combination of location and extracted features such that regions with similar features or locations in a WSI are grouped into the same cluster.

The extraction of features and construction of WSI graph representations by TIAToolbox can be easily integrated with code for training a GCN. The modular nature of TIAToolbox allows for easy integration into a Jupyter notebook as part of the toolbox examples to successfully reproduce the SlideGraph+ results obtained in the original Lu et al. paper [62] using ImageNet deep features and HoVer-Net derived cellular morphology features. TIAToolbox also enabled the ER status prediction using the SlideGraph+ method-
ology. For a full breakdown of these results and a comparison with the original results, refer to Supplementary Table 5. Here the results that we report are obtained using five-fold cross-validation.

2.7 Discussion

TIAToolbox aims to ease the handling of WSI data for analysis and visualization by providing an easy-to-use API that enables seamless reading, pre-processing and analysis of digital pathology slides. Therefore, we hope this will enable the users of TIAToolbox to access a wide and comprehensive set of tools, enabling them to focus more on model development.

Despite the rapid advancement of CPath, there has been no unified software library tailored towards the large-scale batch processing and analysis of pathology slides using state-of-the-art DL models. Previous packages have focused on a smaller subset of features, such as stain normalization or WSI reading. As can be seen in Table 2.7.1, TIAToolbox is an extensive library in terms of the number of features that it supports and therefore, we hope that CPath users will choose to use it for various applications in CPath.

We demonstrate the utility of TIAToolbox by using its core constituents to reproduce the results of two state-of-the-art AI pipelines in CPath. First, we predict the status of molecular pathways and key mutations in colorectal cancer and then we predict with SlideGraph+ the HER2 status from H&E-stained histology images. These pipelines have been implemented in the form of interactive note-
### Table 2.7.1: Comparison of features available in different histopathology image analysis focused software packages.

An exclamation mark (!) indicates a feature that may be partially implemented or is possible with the software, but either requires training or is not directly integrated with the software package. A question mark (?) indicates that there may be an appropriate metric, but no reported value could be found or support for this feature is unclear.

<table>
<thead>
<tr>
<th>Feature</th>
<th>TIA Toolbox</th>
<th>Histo Carto</th>
<th>HEAL Graphy</th>
<th>QuPath</th>
<th>Path ML</th>
<th>CLAM Multi-Scale Tools</th>
<th>stainlib†</th>
<th>IBM CODAIT‡</th>
<th>Histo QC</th>
<th>Histomics</th>
<th>Cytomine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Python</td>
<td>Python</td>
<td>Java (+Groovy)</td>
<td>Java (+Python)</td>
<td>Python</td>
<td>Python</td>
<td>Python</td>
<td>Python</td>
<td>Python</td>
<td>Java/Python</td>
<td></td>
</tr>
<tr>
<td>Modular Usage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Unit Test Coverage</td>
<td>&gt;99%</td>
<td>88%</td>
<td>0%</td>
<td>87%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>?</td>
<td>72%</td>
</tr>
<tr>
<td>Read Visual Fields (PNG, JPEG)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Read TIFF (SVS, SCN, NDPI)</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Read JP2 (Omnyx)</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Read OME-TIFF</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Read DICOM WSI</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Read Zeiss CZI</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Read Olympus VSI</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Fluorescence Imaging</td>
<td>!</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Physical Unit Based Reads</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Tissue Masking / Detection</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Patch Extraction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Stain Separation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Stain Normalization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Stain Augmentation</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Patch Classification</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Tissue Semantic Segmentation</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>!</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Nucleus Detection</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Nucleus Segmentation</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Nucleus Classification</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Deep Feature Extraction</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Graph Learning</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Whole Slide Classification</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Graphical User Interface (GUI)</td>
<td>✓†</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

†stainlib source at [https://github.com/sebastianffx/stainlib](https://github.com/sebastianffx/stainlib), ‡IBM CODAIT deep-histopath source at [https://github.com/CODAIT/deep-histopath](https://github.com/CODAIT/deep-histopath). ✓TIAToolbox includes a basic web-based UI for displaying WSIs on top of each other with adjustable opacity, but not a full-featured GUI.
books, which can be opened and evaluated on cloud platforms such as Google Colab and Kaggle. This highlights how the toolbox can be used to substantially simplify previously complex approaches in CPath. We hope that the examples provided will motivate others to integrate the tools provided by TIAToolbox into their pipelines and help accelerate the development of new methods in CPath. The design of the toolbox ensures that the API remains consistent and easy to use when introducing additional models and tools. The two presented pipelines are algorithmically different, but due to the modular nature of the toolbox, code segments could easily be shared between each method, as highlighted in Figure 2.6.1 where we observe that the first few steps are common due to the re-use of TIAToolbox modules. Both pipelines can also share the same model inference code as highlighted in the figure. For example, all pipelines that use WSIs as input will use our advanced image reading functionality that supports a wide range of WSI formats, including JP2 and those supported by OpenSlide. Also, batch processing and patch aggregation are handled behind the scenes in both pipelines, without exposing unnecessary detail to the user.

We stress that TIAToolbox is not limited to the above two tasks and despite it being desirable to use our toolbox within all steps of a CPath pipeline, this is not a requirement. Due to its modular and extendable design, individual steps and various utility functions can be used in isolation for a broad range of applications in CPath. This helps in training new customizable algorithms on top of existing work. For example, dividing WSIs into patches before aggregating
results is a widely used approach in CPath and this procedure is fully handled within the toolbox. Therefore, any pipeline that involves patch-level processing will benefit from the functionality that we provide. In fact, any patch prediction or segmentation model, based on PyTorch, can be integrated because our API is consistent irrespective of the model choice. The toolbox is not limited to the pre-trained models that we provide. Any model trained outside our toolbox can be seamlessly integrated. We have demonstrated this flexibility with the help of a notebook (see Example Notebook 07) that uses natural images from the ImageNet data set. This enables one to utilize our toolbox for a large array of tasks in CPath, such as cancer staging [88], cancer sub-typing [25], survival analysis, and the prediction of additional molecular pathways [25]. Additional tools can also be leveraged, such as efficient patch extraction, tissue mask generation, visualization and stain normalization, which can all be important steps in the automated analysis of WSIs.

TIAToolbox is available as a PyPi package (via `pip install tiatoolbox`), conda-forge (via `conda -c conda-forge install tiatoolbox`) package, and as a Docker container via the GitHub container registry (docker pull ghcr.io/tissueimageanalytics/tiatoolbox:latest). TIAToolbox is an open-source project, to which additional pre-trained models and features will continue to be added. In future, we will extend the currently available models by training on new datasets, increasing the number of applications for our toolbox. A logical extension would be to train and provide patch prediction models for colon cancer grading[90] and tumour de-
tection in additional tissue types. We also aim to provide instance segmentation, detection, and classification models for tissue structures such as glands, blood vessels and nerves, enabling the extraction of further interpretable morphological features for downstream analysis such as linking these features to survival or investigating spatial profiles of the tumour microenvironment (TME). To enable a better understanding of how models are interpreting images, we aim to include tools that enable the visualization of model activation maps on images. This can be done via techniques such as class activation maps (CAM) [91]. Currently, our SlideGraph+ pipeline utilizes functionalities from various parts of the codebase and integrates them into a notebook. In future, we plan to fully integrate a graph predictor engine within the toolbox, in addition to our existing patch predictor, semantic segmentor, and nucleus instance segmentor engines. Going forward, TIAToolbox could act as an enabler for commercial growth and encourage the use of CPath applications in a clinical setting. We anticipate and encourage users to contribute new features and integrate the provided tools into their own CPath pipelines to accelerate the development of CPath as a field.

2.8 Data Availability

All datasets analysed during the production of TIAToolbox, except for one private oral dysplasia cohort dataset for HoVer-Net+, are publicly available. They can be accessed for research and non-commercial use at the following web addresses: The Cancer Genome

The private oral dysplasia cohort dataset is not available because we do not currently have ethical approval to share this dataset publicly but the trained model is already published with ethical approval details listed in the original publication [79].

2.9 Code Availability

All source code for TIAToolbox is available on GitHub (https://github.com/TissueImageAnalytics/tiatoolbox/tree/publication) and Zenodo[96] (https://doi.org/10.5281/zenodo.6808365) under the BSD 3-clause license. Model weights downloaded at runtime are publicly hosted and maintained on TIA Centre servers under a creative commons non-commercial use licence (CC-BY-NC 4.0). All parts of the toolbox, including model weights, may be freely used for research and non-commercial purposes. Model weights can be made available for commercial use on request depending on ethical approvals from the data source.
2.10 Chapter Summary

TIAToolbox is an open-source library designed for advanced tissue image analytics in computational pathology. It provides a unified framework for reading whole slide images (WSIs), extracting patches, performing stain normalization and augmentation, and running deep learning models for patch classification, semantic segmentation, nucleus instance segmentation, and nucleus classification. The toolbox aims to simplify and streamline the development of analytical pipelines for researchers in the computational pathology community.

The main contributions of TIAToolbox are its comprehensive feature set, support for a wide range of WSI formats, modular structure, reproducibility of published state-of-the-art methods, ease of use, and cross-platform compatibility.

Many of tools provided by the toolbox from image reading to robust pan-tissue nucleus segmentation and classification are foundational to building modern complex CPath pipelines. TIAToolbox can effectively jump-start a CPath project and enable users to get started with reading images and extracting features in just a few lines of code.

The chapter also demonstrates the utility of TIAToolbox by reproducing two full state-of-the-art pipelines: predicting the status of molecular pathways and key mutations in colorectal cancer using a two-stage patch-level classification model and predicting the HER2 status in breast cancer using the SlideGraph+ graph neu-
ral network-based model. The toolbox simplifies these pipelines by providing many necessary functions. This demonstrates how the toolbox can reduce the effort required for implementation and enabling researchers to focus on model development.

The toolbox has addressed common challenges in computational pathology, such as handling multiple WSI formats, efficient image reading, and customization of models. Since its initial release and publication, TIAToolbox has been downloaded over 130,000 times. It has also received community engagement via GitHub stars, issues, and pull requests. The authors have also received positive user feedback at conferences such as ECDP 2022 and 2023.

In summary, TIAToolbox is a comprehensive and user-friendly library for advanced tissue image analytics in computational pathology. It provides a unified framework with a consistent and well-documented API for handling and analyzing WSIs. Its modular design allows for easy integration of new models and customization. By simplifying the development of analytical pipelines, TIAToolbox helps to accelerate the progress of CPath research.
Chapter 3

Whole Slide Image Conversion

In this chapter, we will explore the reasons and challenges of converting whole slide images (WSIs) into more user-friendly, fast, and memory-efficient formats using a Python application. We will first discuss the requirements and formats of WSIs and the motivations behind converting them. Then, we will delve into the implementation of our conversion tool, Whole Slide Image Converter (WSIC), and investigate its impact on deep learning inference using the example case of nucleus segmentation and classification.

3.1 Whole Slide Image Formats

Storing and transmitting Whole Slide Images (WSIs) can be difficult due to their high pixel dimensions and large file sizes, which can range from hundreds of megabytes to several gigabytes. Ad-
ditionally, typical image formats are not ideal containers for slide image data. To address these challenges, specialized formats have been developed specifically for whole-slide imaging to meet the needs of pathologists, scanner vendors, and computational pathology researchers.

Common image formats used on the web and for digital photography cannot support the requirements of digital pathology whole-slide imaging. For example, JPEG file interchange format (JFIF) and extensible image file format (Exif) files which are both commonly referred to as a ‘JPEG image’, are incredibly widely used image formats on the web and by digital cameras. The popularity of these formats is in part due to the low computational complexity of the JPEG algorithm which allows compression to run quickly on digital cameras and the high compression ratio achieved by exploiting properties of the human visual system [97]. However, one major limitation of JFIF/Exif JPEG compressed images is that they are limited to $2^{16} - 1$ (65,535) pixels in width or height\(^1\), which is not sufficient for some larger WSIs.

### 3.1.1 Requirements & Use Cases

When it comes to whole slide images, different stakeholders and use cases have varied requirements. Generally, WSI formats serve three primary purposes:

**Viewing & Clinical Use** Fast interactive panning and zooming

---

\(^1\)The width and height are stored as two-byte unsigned integers in the ‘APP0’ header section of a JFIF file [98].
are the primary requirements for pathologists. Transmission of
data for viewing should also be considered.

**Authoring & Industry** Creation of WSI files is of concern to scan-
ner hardware, software, and researchers. Scanners need to se-
rialize data quickly as it arrives from the sensor. If the writing
of WSI data is slow, this can be the bottleneck for slide scan-
ning throughput. In many computer systems, I/O speed is
the primary bottleneck rather than compute power or memory.
Software needs to be able to decode the image data to display
or process it for use in practice and for research purposes.

**Automated Analysis & Research** Deployed in-production algo-
rithms require high-speed reading of image data to keep mod-
els fed with data as fast as it is processed to maximize re-
source utilization and reduce latency. Furthermore, it is com-
mon for model training, high throughput inference, and other
large-scale data processing to be formed by a distributed sys-
tem such as PyTorch (*torch.distributed*), TensorFlow (*tf.
distribute.Strategy*), Dask, Ray, MLFlow, and many more.
In these systems, data must be sent from storage to each node
for processing. WSI data may need to be formatted in order to
be easily interoperable with distributed software and maximize
system performance.
3.1.2 Image Formats: Containers & Codecs

Image formats and compression algorithms or codecs are closely related and so interdependent that they are often hard to separate. However, when discussing formats, it may be helpful to define what is meant by a container and a codec. Taking JPEG as an example where the word ‘JPEG’ is often confusingly used to mean both the container and the codec. Compressed image data may be stored inside several container formats including JFIF, Exif, tagged image file format (TIFF), and more. Codec, a portmanteau of coder and decoder, describes the method of compressing the image data, in this case, JPEG. This may also be referred to as the compressor/decompressor or algorithm. On the other hand, the container specifies how an image is packaged along with other images and metadata, such as the date of creation, colour depth, and photometric interpretation. The combination of a generic container and a specific codec may be

Figure 3.1.1: Venn diagram showing three primary use cases for WSIs and their overlapping requirements.
called an image format or file type.

3.1.3 Tiles: Random Spatial Access

One very important requirement for WSIs is the ability to randomly access regions of the image. However, many image formats commonly in use, such as entropy coding used by JPEG (either Huffman or optionally arithmetic coding [97]), are unable to accommodate this due to the compression method having backward data dependency. The decompression algorithm, like all programs, can be modelled as a state machine. The backward data dependency means that it is impossible to know what state the program must be in at the start of a region of interest in order to correctly decode the region without first decoding all data preceding this region. This is very inefficient if only a small portion of the image is needed, as is commonly the case when working with WSIs.

To get around this, many WSI formats use tiling. Here, the image is split up into $n \times m$ regularly sized tiles. Each tile is compressed independently. This means that only the tile which overlaps with the required region needs to be decoded. This convenience of being able to decode any part of the image without decoding the entire image does come at a cost. There is increased overhead when compressing tiles individually. Headers may be repeated within the data for each tile, and statistical relationships between pixels across tile boundaries cannot be exploited when compressing. Furthermore, when decoding a region, all tiles within the region must be stitched together and extra pixels cropped.
An additional benefit of tiles being encoded independently is that they render themselves easy to write multithread or multiprocessing-enabled code which can encode or decode multiple tiles in parallel. While it is possible to parallelize certain parts of certain compression algorithms, it is trivial to do so at an independent tile level.

3.1.4 Pyramids: Multiple Resolutions

To enable random access to multiple resolutions, many WSI formats store additional copies of the image at reduced resolution. This is often called a pyramid because it can be visualized as each of the different resolution copies stacked on top of each other, forming a pyramid shape. This increases the total file size and is a trade-off between the speed of access to multiple-resolution representations and the additional storage space used. Generating the additional resolutions on-the-fly as needed may be too slow for interactive viewing or rapid data processing.

Some image formats such as JPEG 2000 and JPEG XL use another technique for achieving multi-resolution access while also avoiding redundant copies. Both of these formats make use of the discrete wavelet transform (DWT). JPEG XL uses the simplest possible wavelet transform, the Haar transform. Whereas JPEG 2000 uses two more complex wavelets, the Le Gall–Tabatabai (LGT) 5/3 and Cohen–Daubechies–Feauveau (CDF) 9/5, depending on whether it is performing reversible or irreversible compression respectively. By applying this transformation, the input image is split into the four frequency bands low-low (LL), high-low (HL), low-high
(LH), and high-high (HH). These are named to correspond with each split of frequencies in the horizontal and vertical direction that this band represents. By applying the DWT recursively \( n \) times to the LL band, a representation of the image split into frequency bands at different scales is constructed.

The original image can be reconstructed, by applying the inverse DWT \( n \) times. However, if the inverse transform is applied less than \( n \) times, the resulting image resembles and scaled-down version of the original image. This means that reduced resolutions can be accessed simply by not completing all the iterations of the inverse DWT. This method of storing multiple resolutions has the advantage of both creating a good representation for compression, as the high-frequency bands have many near-zero coefficients, and there is no redundancy in creating multiple copies of the image at different resolutions. However, some drawbacks are that it is computationally more expensive and that at high downsample factors, this method produces images which appear over-sharpened.

3.1.5 WSI Formats in Use Today

Many image formats are in use today for whole slide imaging and bioimaging, with software libraries supporting dozens of different formats [54], [64] including:

- Aperio/Leica (.svs, .tif)
- Carl Zeiss (.czi, .zvi)
- DICOM VL Whole Slide Image (.dcm)
Tiles form a (usually) non-overlapping mosaic grid. The size is homogeneous, (usually) even between levels.

Figure 3.1.2: Illustration of an image pyramid made up of five levels, the full baseline resolution and four reduced resolution copies. Each level of the pyramid is also tiled. Auxiliary images may also be included inside a format container, such as a barcode or thumbnail image, in addition to the smallest pyramid level.

- Hamamatsu (.vms, .vmu, .ndpi)
- JPEG XR (.jxr)
- Leica (.scn)
- MIRAX (.mrxs)
- NGFF (.zarr)
- OME-TIFF (.ome.tif, .ome.tiff)
- Omnyx/Inspirata (.jp2, .rts)
- Philips (.tiff, .isyntax)
- Sakura (.svslide)
- Trestle (.tif)
• Ventana (.bif, .tif)

• Generic tiled TIFF (.tif, .tiff)

While there is much diversity in the range of formats there are a few containers and principles which they are all based on. All of these formats use some form of tiling for spatial access, and all use a pyramid or wavelet transform for multi-resolution access. Tiling can vary slightly between formats, with some formats allow for an overlap between tiles to avoid or reduce border effects. However, in most formats, tiles are non-overlapping. Additionally, JP2 (JPEG 2000) images employ multiple levels of hierarchical tiling dividing regions into tiles, precincts, and code blocks.

While there is some variation in containers used for WSI formats, the majority are a variant of the Tagged Image File Format (TIFF).

Tagged Image File Format (TIFF)

The Tagged Image File Format, or TIFF for short, is a popular file format for faxing and storing images or digitized documents. Introduced in 1986 by Aldus Corporation and acquired by Adobe Inc. in 1994, TIFF attempted to unite desktop scanner manufacturers behind a common file format. Initially, in the first public specification, Revision 3.0, it only supported uncompressed binary images due to the limitations of scanners at the time. However, the format soon received several updates adding support for features such as greyscale and full-colour images. Since then, TIFF has become a widely used standard in the graphics industry. It is known for its
ability to support very large images.

In Revision 6.0 of the specification, published 1992 [99], the specification was split into two parts: part one, baseline, and part two, extensions. This laid the foundations for many WSI scanner formats with the introduction of tiled images.

TIFF, as of revision 6.0 including part two extensions, supports both tiling of images and the linking of different resolutions forming a pyramid. Due to support for these two key features, the lack of active patents encumbering TIFF, and the format’s longevity, many WSI formats are based on TIFF as a container. However, because many of the features of TIFF used by WSIs are part of the optional part two extensions specification, it is common for software and libraries to not fully support WSIs. To complicate matters, many WSI formats based on TIFF use non-standard extensions such as custom metadata tags, non-standard compression codecs, or overflowing values when they are too large.

One major development in TIFF was the introduction of ‘BigTIFF’ [100]. As TIFF uses 32-bit numbers for specifying byte offsets within the file for tiles and image file directories (IFDs), a tiled TIFF couldn’t exceed 4 GiB in size [101]. After discussion on the mailing list for libtiff, a major open-source TIFF library, support for 64-bit offsets was implemented by Ole Eichhorn while at Aperio and donated to the public domain forming a variant called BigTIFF [102]. Regular TIFF, also known as ClassicTIFF, and BigTIFF became the container or basis for many whole slide image formats such as Aperio SVS [103], Leica SCN, Hamamatsu NDPI, and OME-TIFF
3.2 Why Convert Whole Slide Images?

Digital pathology involves the use of a range of digital whole slide image formats, with dozens of formats currently in use. There are many reasons to need to convert between formats, including:

- Reduce code complexity and cognitive load by working with one format instead of many.
- Give data predictable performance characteristics.
- Allow for a user to trade-off between disk space and decode speed.
- Enable compatibility with software tools.
- Avoid requiring proprietary software and licensing by converting to an open format.
- Optimize to a format optimized for machine learning applications.
- Convert to a cloud object storage-friendly format for storage in and streaming from a large data lake.
- As part of a de-identification process.
- Enable collaboration where a clinical use format may be required.
3.2.1 Reduction of Complexity Through the Creation of Homogeneous Datasets

Working with many WSI formats at the same time can be difficult to manage. It may require code to be written for many image libraries with differing interfaces. This can increase cognitive load for the programmer, and increase the chance of bugs by introducing many possible code paths. Alternatively, an abstraction layer library such as TIA Toolbox, BioFormats, or OpenSlide, can be used. However, this may introduce many unnecessary dependencies. Furthermore, if the WSI reading library does not support a particular format, a user must implement a special case. The use of one format can be simpler and advantageous in several ways.

By moving the complexity of converting images to the data ingestion pipeline, it enables applications to be much simpler. Using a single format can reduce dependencies to a single library for image reading. This also reduces the number of code paths taken by a program which can make debugging and optimization easier.

Reducing dependencies can be important for environment management systems such as Anaconda which checks for compatibility of dependencies by solving a boolean satisfiability (SAT) problem. This is a well-known hard problem to solve and was the first problem to be shown to be NP-complete, i.e. complex to find a satisfying solution but polynomial time to verify [105]. Reducing the number of dependencies can greatly reduce the time to solve the environment.

Furthermore, by minimizing dependencies there is a reduced chance
of regression issues and a reduced attack surface for bug exploitation or supply chain infiltration. Supply chain attacks have already been demonstrated within the Python package index (PyPI) [106], [107].

Moving the complexity of handling many formats from processing code to the data ingestion or preparation stages of a pipeline allows the code which is processing image data to be simplified. If data is normalized as it is ingested into a large data lake, end users are relieved of the burden of handling many file formats. This is the approach taken with some existing data lakes, such as The Cancer Genome Atlas (TCGA) [108].

3.2.2 Predictable Performance Characteristics

Each format has different characteristics in terms of compression and encode/decode performance. For example, JP2 images typically have a small size on disk due to the good compression ratio of the JPEG 2000 compression codec. However, they are also much slower to decode than other, less compact, formats such as JPEG compressed TIFFs. A heterogeneous dataset of images may exhibit vastly different data decode times between WSIs.

3.2.3 Trade-Off Between Disk Space and Speed

By converting the dataset to a single fast-to-decode but less compact format, a user can trade disk space for a predictably fast decode speed. Decoding speed in particular is important for machine learning tasks in order to keep GPUs fed with data. Furthermore, converting from a slow format is a one-time slow read to enable
subsequent reads to be much faster.

Cost is also a significant factor when considering space and time requirements. It is often stated that storage space is cheaper than both CPU time and paying an employee’s wage. For example, time consumed by handling extra complexity. Original images may be kept long-term and at a lower cost in ‘cold storage’ solutions, such as Amazon S3 Glacier or Wasabi immutable storage, in case they are needed at a later date. Meanwhile, a copy converted to a common format may be kept in the more costly main data lake storage for frequent access.

### 3.2.4 Transitioning to Open Formats

Converting WSIs to an open format avoids the requirement for proprietary software, which can be problematic. Sticking to open software and standards makes interoperability and collaboration much easier, especially within a research group or distributed system where many users or machines would need to be licensed to use proprietary software. Industry vendors commonly want to use their proprietary format, citing reasons such as existing formats either not offering high enough compression and quality or being too computationally intense to allow for streaming data from a digital slide scanner to a file.

Exporting from proprietary formats to open formats such as TIFF or DICOM is often an option. However, it is usually not the default format used by vendors. Converting to a more open format may reduce the quality of the output image due to compounding
non-reversible ‘lossy’ encoding errors.

Notably, some codecs used in file formats have both open and closed implementations, such as JPEG 2000. Proprietary software such as Kakadu can offer increased performance or additional features. On the other hand, open tools can be used to increase accessibility to the field, which is good for education and research.

It should also be noted, there is not yet a clear single dominant open standard with digital pathology. OME-TIFF, DICOM, and NGFF are all currently competing open standards.

Finally, converting to an open format can avoid vendor lock-in and becoming dependent on a particular set of proprietary software and hardware tools. It is possible that a vendor may go out of business or become acquired by another, such as when GE Omnyx was acquired by Inspirata in 2018 [109]. As part of this process, tools may become discontinued and access to software, APIs, or licensing required to decode the image can be lost.

### 3.2.5 Cloud Optimization

Some formats may not be optimized for big data cloud object storage. Serial formats such as TIFF and DICOM are single large binary blobs and are thus not streamable in chunks from serverless object storage solutions, such as Amazon S3. Whereas Zarr-based formats such as NGFF are composed of many independent object chunks which are well suited to server-less solutions. Serial formats such as DICOM, JP2 and TIFF can be streamed, via server protocols such as DICOM server and JFIF. However, as these require a server to be
running, they are more costly. Additionally, they are more complex to set up and harder to scale under unexpected load.

3.2.6 Clinical Import and Export

Files may contain patient-identifiable information. This is especially common with DICOM, as it was designed to hold patient data. Converting by copying just the pixel data and essential metadata to another format is a way to ensure that metadata is fully de-identified when ingesting clinical data.

Conversely, when collaborating with clinicians, it may be necessary to convert to a clinical use format which is compatible with a picture archiving and communications system (PACS).

3.3 Challenges for WSI Conversion

When implementing code for the conversion of WSIs there are several significant challenges to overcome including managing available memory, dealing with complexity, correctly transferring metadata, and taking advantage of differing optimal read and write chunk sizes.

Firstly, hardware configurations typically do not have enough available memory to hold an entire decompressed whole slide image array. Therefore, processing the image in a streaming manner, a chunk at a time, is preferable. This is the approach taken by other tools such as libvips. Doing so also allows for parallel processing of WSIs which can achieve better CPU utilization. This is because most workloads are I/O bound, especially in a GPU compute or
distributed system. Data fetching can be performed in parallel with other I/O and computation. This alleviates bottlenecks and allows for higher CPU utilization and overall throughput.

Another important challenge is handling the complexity of many distinct file formats. In addition to there being many WSI formats in use, there are subtleties of each format which must be taken into account and deviations from a standard specification are common. For example, a valid TIFF file must have a tile size which is a multiple of sixteen in both width and height. However, a DICOM file can have tiles of any size. Additionally, Hamamatsu NDPI TIFF files may have blank zero-length tiles to indicate a background region or may overflow integers when specifying the width and height of the image. Handling special cases for each format and different APIs can lead to a codebase which is buggy, unwieldy, and impractical. By converting to a single format, the code performing analysis of the images can be far simpler, with fewer branching conditions and many fewer possible code paths. This allows for easier testing and code comprehension while reducing the risk of an uncommon path hiding a subtle bug.

Codecs used by some images are very computationally and memory intensive to encode or decode. For example, the JPEG 2000 codec is notoriously computationally intensive [110] and memory hungry. This is especially noticeable with JP2 files that use a single large tile, such as those created by an Omnyx GE (now Inspirata) slide scanner. Decoding these WSIs as many small regions is much slower than decoding fewer larger regions for this format. However,
it may be desirable for the output format to use smaller tiles. Tiled
TIFF files using other compression codecs, such as JPEG or WebP,
perform optimally with tile sizes between 256- and 512-pixels square –
typical tile sizes for SVS and NDPI images. Furthermore, serial
formats such as TIFF require that tiles be written in left-to-right,
top-to-bottom order. As such, some efficient schemes for re-tiling
the image data, i.e. changing the tile size and re-ordering tiles, must
be implemented.

Metadata is another important consideration when converting
WSIs. The capture resolution of the image is critical information
when performing an analysis of such images. This is because many
algorithms and models are tuned to function optimally at a partic-
ular microns-per-pixel resolution. An image may be re-sampled to
the required resolution, but only if the original capture resolution
is recorded within the image metadata. Image formats handle this
metadata differently, and it is easy for it to be lost or corrupted
during conversion.

Lastly, re-encoding image data can significantly impact the per-
ceptual quality of the image. Therefore, it should be avoided or
mitigated where possible. This is because compression codecs are
irreversible or ‘lossy’ functions, with each generation of re-encoding
compounding errors in the image. There is a ‘replicator phenomenon’
or meme\(^2\) culture on the internet with images commonly down-

---

\(^2\)This term for a repeatedly shared idea, phrase, or way of doing something which spreads like a virus or gene was coined by Richard Dawkins in ‘The Selfish Gene’, published in 1976. It is an abbreviation of the Greek word mimeme – meaning that which is imitated. The term was later used more specifically in the context of internet culture by Mike Godwin in the October 1994 issue of Wired magazine during a thought piece.
loaded, optionally altered, and re-uploaded elsewhere. Images which have become significantly corrupted due to aggressive quantization or compounded generational image errors are now so common that they have become a meme themselves with the meme expression ‘needs more JPEG’.

### 3.4 Implementation of WSIC

WSIC was implemented using several open-source image and array libraries including OpenSlide, tifffile, OpenJPEG and Glymur, Pydicom, wsidicom, OpenCV, imagecodecs, Zarr, XArray, Dask, and NumPy. Many of these are optional dependencies and are only imported when required.

The main design goal of WSIC is to make it easier to use from the command line and as a Python package in projects. WSIC does this by providing an easy-to-use command line interface along with clear user feedback via nested progress bars.

A multiprocessing single writer multiple readers (SWMR) design is used (Figure 3.4.1). Reads may be performed safely in parallel as a file can be opened in read-only mode from multiple processes without risk of modification, and thus potential corruption, of the source file. However, writing usually be performed synchronously and cannot be safely parallelized. This is because there is a backward data dependency between encoded tiles. The offset for writing a tile can only be determined when all previous tiles have been encoded as it will be the sum of the length of all previous encoded
Tiles may extend beyond the dimensions of the WSI. These will be trimmed when the region is read.

Figure 3.4.1: Diagram illustrating the SWMR pattern used by WSIC. Large tiles are continuously read from the source and placed onto a queue by each reader process in parallel. This is done asynchronously, and tiles may be placed in the queue out of order. Meanwhile, a single writer process continually checks the tile queue and writes any enqueued tiles to the intermediate before checking if the next required region for writing is available in the intermediate. If available, the next tile region is written to the destination.

The SWMR system is particularly useful when the source image is slow to decode but the destination format is fast to write, for example when converting a large single-tile Omnyx JP2 image to a tiled TIFF compressed with a codec such as JPEG, ZStandard, or WebP.

In order to allow for re-tiling, an intermediate representation is used. This is a lossy compressed Zarr array, kept in a temporary directory (e.g. ‘/tmp’ on POSIX systems). The default Zarr compression is used which combines the meta-compressor Blosc with ZStandard. This offers a good balance between compression speed and compression ratio. This prevents the intermediate presentation from using a very large amount of disk space but is also very fast for reading and writing. By using this intermediate, tiles can be writ-
ten out of order as they are decoded asynchronously by the multiple reader subprocesses. The single synchronous writer can simultaneously continuously write tiles as the next required region is decoded and becomes available. The writer awaits the required region in the intermediate array if it has not yet been decoded and written to the intermediate by a reader process.

3.4.1 Lossless Repackaging

WSIC also implements a ‘transcode’ mode which copies encoded tiles from one container format to another without decoding and re-encoding the image data. This allows for very fast and lossless repackaging or transcoding of the image from one format to another. However, this is limited in that it can only be performed when the tile size and compression codec of the input and output format match. This could be an issue for example when converting from a DICOM to a TIFF file. This is because in TIFF the tile size must be a multiple of sixteen whereas in DICOM the tiles can be any size. Furthermore, it is generally not possible to transform the compressed representation used by one codec into another. There are some exceptions to this. For example, the upcoming JPEG XL standard intends to support lossless repackaging of JPEG at reduced file size. Although at the time of writing this is not in the API for the jxl reference implementation library. Lastly, there are some tools for reversibly altering the representation of JPEG data such as jpegtran, MozJPEG, and Dropbox lepton.
3.5 Conversion Benchmarking

In order to benchmark the performance of WSIC, a set of sample images was converted using WSIC and a range of freely available open-source command line interface tools:

**tiff2jp2** from the Glymur Python package, which is a wrapper around the OpenJPEG library.

**wsi2dicom** a tool from Google Cloud Platform.

**vips tiffsave** from the libvips library.

**bfconvert/bioformats2raw** which are both based on the BioFormats library but support different formats.

GUI programs which use the same library as the command line tools were excluded from the comparison. For example, OMERO NGFF-Converter was excluded as it is also a user interface for the BioFormats library.

Conversions were measured using a desktop computer with a six-core 3.00GHz Intel Core i5-8500 processor and 64 GiB of DDR4 memory. Input formats were JP2, SVS, TIFF, and DICOM. Output formats were JP2, SVS, TIFF, DICOM, and NGFF. NGFF was used as output only because it is a relatively new format where both the format and tooling are still evolving. Except for SVS, where a thumbnail must also be generated, each conversion was performed using only the full high-quality resolution image. This was to reduce confounding factors from the generation of downsamples, where implementations may have involved decisions that trade-off between
Figure 3.6.1: Bar chart showing conversion speed in megabytes per second across the compared open-source CLI conversion tools. The times are an average across several images in the benchmark suite. The fastest conversion for each Input → Output pair is highlighted with a hashed bar.

quality and performance. Furthermore, for most output formats, additional resolutions may be efficiently appended after initial conversion using a chosen downsampling method.

When comparing tools, parameters were normalized across tools where possible. The number of worker sub-processes was set to six (the number of CPU cores available) and the output tile size was fixed at $512 \times 512$ pixels.

### 3.5.1 Bulk Repackaging of TCGA SVS Slides

In addition to the main conversion benchmark test, small batch conversion of 20 TCGA SVS images (22.8 GiB in total) across ten tissue types was performed using GNU parallel [111] and WSIC ‘transcode’ mode to estimate the time required to batch convert the whole of TCGA (11.8 TiB) using this method.
3.6 Conversion Benchmarking Results

Support for reading and writing varied between tools, with WSIC supporting the writing of all formats tested (generic tiled TIFF, SVS, JP2, DICOM (‘.dcm’), and NGFF. BioFormats CLIs, bfconvert and bioformats2raw, also supported writing all formats tested between them. However, half of the cases when reading or writing a JP2 file with bfconvert failed due to an out-of-memory error. The tiff2jp2 tool only supported conversion from TIFF to JP2, while vips only supported writing to TIFF. Additionally, conversions with tiff2jp2 exited with a non-zero exit code and missing tiles were filled by black regions at the bottom and right edges. WSIC preserved the resolution metadata in all tests where the output supported standard resolution metadata, whereas other tools frequently lost this metadata during conversion.

Conversion rates for WSIC and other tools are shown in Figure 3.6.1. The fastest time for conversion was achieved by WSIC in all conversions except for DCM to JP2, SVS to DCM, and TIFF to DCM where bfconvert, wsi2dcm, and wsi2dcm respectively were faster. However, it should be noted that bfconvert failed to write larger JP2 files. Furthermore, WSIC outperformed wsi2dcm in conversion from SVS/TIFF to DCM when the ‘transcode’ mode is used.

Batch SVS transcode for the sample of 20 slides required 135 seconds in total. Extrapolating this to all diagnostic FFPE slides in TCGA (10.8 TiB) would require 23 hours, using the same single desktop computer as in the conversion benchmark.
3.7 The Impact of Image Compression on Deep Learning Inference

Storage and transmission of large WSI files are made far more accessible by the use of image compression methods, with JPEG and JPEG 2000 being the widely used WSI codecs. For example, both are used for Aperio SVS images in The Cancer Genome Atlas (TCGA), a common data source in the field of CPath. The majority of image compression algorithms act as irreversible one-way functions, also known as ‘lossy’ compression. As such, they introduce errors, or visible compression artefacts, into images.

The nature of compression artefacts is characteristic of the image codec used. For example, JPEG exhibits distinctive blocking artefacts, especially at low-quality parameter settings. This is due to compressing $8 \times 8$ regions independently. This block-wise approach allows the algorithm to be easily parallelized and perform well on low powered hardware but also increases discontinuities at block boundaries. JPEG compression additionally exhibits distinctive colour distortion effects due to chroma subsampling and aggressive quantization at low-quality settings. With high-quality codec parameter settings, perturbations are typically limited to high-frequency noise, which imperceptibly impacts the human interpretation of the image, often referred to as ‘near lossless’ compression. Other codecs exhibit different perceptual artefacts.

Some work has examined the impact of JPEG and JPEG 2000 compression on CNN performance [112]–[114]. Doyle et al. [114]
initially explored the impact of JPEG and JPEG 2000 compression on the segmentation of large structures such as glands and lumen within WSIs. Chen et al. [114] investigated further by examining smaller slide objects, namely cell nuclei. However, none of these works also looked at performing multi-class nucleus classification. This is important to note, as cell type inference may be informed by small-scale features such as intracellular features such as the presence of granules within the nucleus. This kind of high-frequency information is the most likely to be impacted by compression. Additionally, segmentation has been limited to binary classes. Multi-class problems are typically significantly more difficult, with a much greater scope for confusion between classes. In this work, we instead explore a multi-class tissue semantic segmentation task.

Furthermore, previous work has been limited to JPEG and JPEG 2000. While these are currently common compression codecs, there are other codecs which may see adoption in digital pathology such as JPEG XL which is beginning to see adoption in software libraries and web browsers. Additionally, with the DICOM VL Whole Slide Imaging standard becoming more common, other DICOM standard codec such as JPEG-LS may be used.

Despite the widespread use of compression in digital whole slide imaging and computational pathology, it remains unclear if deep-learning models are robust enough to lossy image compression schemes. This may be especially relevant if models have not been trained on images compressed using a particular codec and quality setting [113].
3.7.1 Method

We have analyzed the processes of nucleus instance segmentation and nucleus classification across a range of image qualities. The dataset for nucleus segmentation and classification contains twenty-five images \((n = 25)\) across five common tissue types:

- breast
- bladder
- colon
- lung
- kidney

Each image is a 10,000×10,000 pixel extract from a TCGA SVS image created using Aperio ImageScope and saved using lossless compression.

Deep learning models for these tasks were run using TIA Toolbox [115]. The HoVerNet [61] model trained on the PanNuke dataset [84] was used for both nucleus instance segmentation and classification.

A range of codecs were selected for comparison:

**JPEG 1** The first JPEG codec released, also known as just ‘JPEG’.

**JPEG 2000** Specifically the irreversible/lossy mode.

**JPEG-LS** Not to be confused with from lossless JPEG 1\(^3\) or lossless JPEG XT\(^4\).

\(^3\)The JPEG 1 specification defines a lossless mode. However, this is rarely implemented and supported.

\(^4\)JPEG XT is a backwards-compatible extension of the original JPEG 1 specification with additional features including floating-point representation and lossless compression.
JPEG XL A new codec for which the version 1.0 specification is complete. The reference implementation is still in active development, with v0.8.2 as the current version at the time of writing [116].

To create the images using a variety of image compression codecs, WSIC was used to convert the SVS image to an NGFF Zarr image contained within an SQLite database file. An SQLite container was selected to reduce file-system churn and overhead from many small files that would be created using a Zarr directory or zip store. TIFF was not used as a container format due to limited support for codecs outside those defined in TIFF specification via the tifffile library.

Default parameters for codecs were used other than the quality or ‘level’ setting. WSIC uses the imagecodecs library for encoding image tiles. The imagecodecs library depends on a dominant implementation of each codec including libjpeg-turbo, OpenJPEG, CharLS, and libjxl for JPEG, JPEG 2000, JPEG-LS, and JPEG XL respectively. In each case, this is either the reference implementation or a widely used de-facto open implementation. The ‘level’ or quality parameter corresponds to a different target metric specific to each codec. The selection of values for this setting was made to correspond with an approximate PSNR range of thirty to fifty decibels, or to the highest possible quality in the event that a PSNR of fifty decibels could not be achieved.
3.7.2 Evaluation Metrics

As in the paper by Chen et al. [112] and Zanjani et al. [113], the pixel-wise F1 (Sørensen-Dice) score was used for the evaluation of nucleus segmentation.

For evaluating nucleus classification, the deviance in the count for each cell type from the original image was measured to assess the change in cellular composition.

3.7.3 Inference Results

Nucleus Instance Segmentation

Results for nucleus instance segmentations for JPEG 1 and JPEG 2000 are similar to those found by Chen et al. [112] with JPEG performing significantly worse than JPEG 2000 at the same compression ratio (Figure 3.7.1 A). JPEG XL performs slightly better than JPEG 2000 at lower compression ratios on average, although within the 95% confidence interval for JPEG 2000. However, the difference becomes insignificant above a ratio of 0.125.

Nucleus Classification

Across all codecs tested, nucleus classification predictions remained significantly inconsistent with the original image below a PSNR of 40 dB (Figure 3.8.1). JPEG 1 in particular showed large variation across all nucleus types across the entire range of PSNR and compression ratio (Figure 3.8.2). Even at the highest quality setting of 100, JPEG 1 images resulted in significantly underpredicted inflam-
Figure 3.7.1: A PSNR versus pixel-wise F1 (Sørensen-Dice) coefficient for nucleus segmentation. B Compression ratio versus pixel-wise F1 score. The shaded area indicates the 95% confidence interval.
matory and dead cells and significantly overpredicted connective and neoplastic nuclei.

Inflammatory cell prediction was the most stably predicted cell type across the range of PSNR and compression ratios. Prediction of dead cells was the most variable class. At low PSNR values, JPEG-LS predictions were the closest to the original image. However, JPEG-LS was the only codec to not achieve a compression ratio under 0.15.

Across all classes and codecs, predictions appear to become consistent with the original image for a PSNR greater than 40 dB.

3.8 Chapter Summary

This chapter discusses the conversion of whole slide images (WSIs) and the impact of image compression on deep learning inference. The main findings are as follows:

- Converting WSIs to a single format simplifies pipeline development code, reduces dependencies, and can improve performance when using slow to decode image formats.

- The WSIC Python application is a helpful tool for WSI conversion, with faster conversion speeds compared to other open-source tools.

- WSIC can be used to optimize storage and computational resources, improve interoperability, and support data normalization and collaboration in digital pathology.
Figure 3.8.1: Peak signal-to-noise ratio (PSNR) in dB versus the percentage of detected nuclei as compared with the original image. A subplot is shown for each class of nucleus detected by HoVerNet trained on the PanNuke dataset. The shaded area indicates the 95% confidence interval.
Figure 3.8.2: Compression ratio versus the percentage of detected nuclei as compared with the original image. A subplot is shown for each class of nucleus detected by HoVerNet trained on the PanNuke dataset. The shaded area indicates the 95% confidence interval.
• Image compression can significantly impact the performance of deep learning models, with different codecs and quality settings affecting model accuracy. Targeting a PSNR of at least 40 dB can minimize the impact for the fine detail tasks of nuclear instance classification.

In conclusion, this chapter provides valuable insights into WSI conversion and the importance of considering image compression in deep learning inference. It highlights the benefits of using a single format, introduces the WSIC tool, and raises considerations for model development and deployment in digital pathology. Further research is needed to validate and expand on these findings.
Chapter 4

Leveraging Alternative Representations of Histology Images

4.1 Leveraging Self-Supervised Representations

Determining the cellular phenotypic composition in different parts of a multi-gigapixel WSI can be very useful for downstream analysis in computational pathology for quantitative profiling of tumour heterogeneity. For example, the co-habitation of cell types such as tumour and immune cells, known as tumour infiltrating lymphocytes (TILs), is a crucial indicator of immune response to a malignant tumour [117], [118]. Automated prediction of cellular composition in histology images is a challenging task due to the large intra- and inter-class variability in the appearance of different types of cells.

Existing methods for the prediction of cellular composition can be
broadly classified into two categories: strongly supervised segmentation-based methods and weakly supervised regression-based methods (Figure 4.1.1). Segmentation-based methods may be unsupervised using classical image processing techniques such as threshold [30], active contour, or level set methods. They may also be neural networks trained in a strongly supervised manner [95], [119], [120] using full pixel-level segmentation masks. These masks may be created entirely by hand or generated via computer assisted methods with basic manual human input. Methods for this include the watershed algorithm, generalizations of and improvements on the watershed method [121], or neural networks which can expand simple point or scribble annotations into full segmentation masks [58], [122]. However, the requirement of pixel-level annotations for training such methods and their computational complexity can be a bottleneck in their large-scale applicability.

To overcome the dense annotation bottleneck, we cast the prediction of cellular composition as a regression problem. A regression model can be trained directly on weaker labels of counts of various types of cells in an image, as opposed to pixel-level annotations as required by segmentation models. Such regression-based methods have been designed to predict bacterial and embryonic cell counts from light fluorescent microscopy images using density maps [123] and manifold learning [124] respectively. However, the development of such weakly-supervised regression models for cellular composition analysis in routine histology images has seen limited progress. Zheng et al. [125] proposed a manifold regularized network for cell
Figure 4.1.1: Concept diagram of the proposed CellCoRe method (top) for predictions of counts of different types of cells in a histology image. Compared to a typical instance segmentation and classification-based method (bottom), our method uses weaker annotations of count labels and a generic unsupervised feature extractor that requires no annotations.

Localization and counting in histology and microscopy images. Cohen et al. [126] demonstrated training a regression network and using an overlapping stride to produce a cell density map for localization. However, these methods are limited to homogeneous cell localization and predict counts of single types of cells only.

We present CellCoRe (Cell Count Regression) model that employs unsupervised representation learning on domain-specific data with no annotations and uses that representation to predict cell composition in a histology image using weaker count labels. We show that generic unsupervised feature representations used by CellCoRe compare favourably to the state-of-the-art task-specific nuclei instance segmentation and classification networks without requiring instance segmentation masks for training. The use of such generic representations which can make use of the abundance of unlabelled
histology data has many potential applications. This work demonstrates a non-trivial application of such features. In addition to producing good predictions with weaker labels, CellCoRe also uses fewer parameters than other high-performing methods. Lastly, the regression head is very fast to train. Using the same pre-trained feature extractor, this methodology could be applied to other H&E image analysis tasks with a rapid turnaround or serve as a useful baseline for future work.

4.1.1 The Proposed Method

The proposed method (shown in the top half of Figure 4.1.1) consists of two main parts: unsupervised representation learning using a CNN as a pre-training step, followed by the training of a secondary regression head which uses features from the CNN.

Unsupervised Representation Learning

Unsupervised representation learning produces a feature embedding for an input image that captures important primitives associated with images in computational pathology. This enables the downstream regressor to accurately model the cellular composition prediction problem.

A large dataset of 2.29 million unlabelled H&E images was compiled for this unsupervised representation learning. This is nearly twice the size of the well-known ImageNet dataset [127] which contains 1.28 million images. This dataset was constructed by extracting square RGB image crops, also known as patches, from 3,277
H&E stained diagnostic WSIs obtained from The Cancer Genome Atlas (TCGA) across 22 different tissue sites including bladder, breast, colon, and kidney. All patches were extracted with dimensions of $256 \times 256$ pixels. To avoid sampling many regions which do not contain any tissue, simple Otsu’s threshold [30] tissue masking was used to eliminate background regions.

Additionally, the microns-per-pixel resolution of the WSI images varied between 0.16 and 0.5 due to the many image sources in TCGA. During patch extraction, the resolution of the images was normalized to 0.5 microns-per-pixel according to the resolution metadata reported by OpenSlide [64].

**SwAV Training**

Unsupervised representation learning was performed using Swapping Assignments Between Views (SwAV) [63] and PyTorch [77] (version 1.5.1). As an unsupervised training step, SwAV takes an image patch as input and learns to generate a corresponding prototype vector representation of the patch that is invariant under various domain-specific augmentations. Multiple augmented copies of an input image, known as views, are generated. For this application two views were used as this was shown to work well in experiments on natural images by Caron et al. [63]. These augmented views are then passed through a convolutional neural network, such as ResNet50 [75], to generate a feature representation and cluster assignment for each view. The network is trained to ensure that the multiple augmented views of each input image are assigned to
the same cluster. In testing, SwAV assigns a given input image to one of \( n \) prototype vectors, which act as its soft cluster assignment. In our application, a value of \( n = 3,000 \) was used for the number of prototype vectors. This unsupervised representation learning method requires no labelling of the input data and has been shown to work well on large and unbalanced datasets when these unsupervised learned weights are used for feature extraction or as model initialization for a downstream task such as image classification [63]. The use of the aforementioned large-scale histological dataset in SwAV training ensures that the resulting cluster assignments are based on domain-specific invariances under augmentations such as random resized crop, horizontal flip, vertical flip, colour jitter, and blurring. Since SwAV is a general learning scheme which is agnostic of the underlying CNN architecture, we used ResNet50 [75] as the backbone network in SwAV. This model offered a balance between the number of parameters and model performance. The network was trained for 225 epochs over this dataset using a batch size of 128. Two Nvidia Tesla V100 GPUs were used for training.

4.1.2 Evaluating the Learnt SwAV Representation

After training the SwAV model, the utility of the embeddings and cluster assignments was examined. To do so, the external validation set of 7,180 tissue patches from Kather et al. [94] for colorectal tissue classification was used. All the patches were passed through the SwAV model to obtain a 3,000-element representation vector for each patch. Instead of applying the argmax to obtain a cluster
Figure 4.1.2: Scatter plot showing UMAP [128] embeddings, computed from the trained SwAV model outputs, of the Kather et al. [94] validation set. The points are coloured with ground truth labels shown against the colour bar on the right.

Assignment, these embeddings were reduced to a two-dimensional embedding using uniform manifold approximation (UMAP) [128]. By plotting the SwAV UMAP embeddings and colouring them with the ground truth labels, it can be seen that the SwAV model learnt a strongly discriminative representation. Figure 4.1.2 shows clear clusters in the plot which correspond with the ground truth labels.

In the cluster of mucus (MUC) patches, there is one patch which is labelled as normal (NORM). Figure 4.1.3 shows this lone NORM patch and the eight nearest neighbours in the UMAP space. Here the normal patch has a similar appearance to the mucus patches and can be hard to differentiate without a larger context. This demonstrates that these learnt representations have the potential to
Figure 4.1.3: A single patch labelled as normal (NORM), and the eight nearest mucus (MUC) patches in the UMAP embedding space.

detect labelling errors in supervised datasets.

4.1.3 Nuclear Composition Prediction

Accurately predicting nuclear composition can be difficult because it is usually approached as a multi-stage problem requiring localizing cell nuclei, discriminating overlapping or touching cells, and classifying cell types to obtain a histogram of cell type frequencies for an image. This has previously been achieved by specialized network architectures such as HoVerNet [61], which comprises a backbone of residual units and three decoder heads for predicting horizontal and vertical local nuclei coordinate maps, a nuclear pixel mask, and a cell classification map. For many downstream tasks, only the frequency of each cell type is relevant and a large complex network for predicting segmentation masks is unnecessary. This work shows
that cell type counts can be directly predicted with performance competitive with HoVerNet. This is done using a common convolutional neural network architecture and pre-training via unsupervised representation learning.

Data Exploration & Pre-processing

In preparation for training a projection head for regression on top of the SwAV pre-trained ResNet network, an investigation was conducted to explore the representation of cellular composition. To determine if the learned representation could be useful in predicting cellular composition, labels from the PanNuke dataset [84], [85] were utilized. These labels consisted of pixel-wise instance segmentation masks with a class label corresponding to each channel, as well as an additional channel for a background versus nucleus map.

For each image, the corresponding cell segmentation mask and classification labels were transformed into a count vector of the cell types by counting the number of unique non-zero values in the channel of the instance mask corresponding to the cell type.

The set of count vectors was then reduced to a three-dimensional space via UMAP and assigned an RGB colour. Additionally, each image in the dataset was passed through the SwAV trained ResNet to obtain a feature vector. These feature vectors were also reduced to a three-dimensional space via UMAP and plotted as a scatter plot, with each point assigned the corresponding RGB colour assigned to the label vector. The aim was to assess if there was a similarity in structure between the locations of the UMAP-embedded SwAV
Figure 4.1.4: Scatter plots of UMAP embedded SwAV features for the PanNuke dataset. Points are coloured using an RGB value derived from a UMAP embedding of the labels which are vectors containing the count for each cell type. Each quadrant of the figure shows the same point cloud from a different perspective.

It can be seen from the resulting scatter plot (Figure 4.1.4), that there is some clear colour grouping with regions of mostly red, green, or blue points. This indicates that the SwAV features are likely to be useful for predicting the cell count vector labels.
CellCoRe Regression Model Training

The objective of nuclear composition prediction is to train a predictor to produce a vector $y_i = f(x_i; \theta)$ corresponding to cell counts of individual cell types in a given the SwAV image patch, which is represented by its feature representation $x_i$. To directly predict cell counts, the ResNet50 model trained using SwAV is used as a feature extractor. These 128-dimensional ResNet50 features are then used as inputs to a regression model trained on the publicly available PanNuke dataset. Two regression models were evaluated using scikit-learn [73] (version 0.22.1), ordinary least squares (OLS) and a multi-layer perceptron (MLP) with the MLP giving better predictions in all cases. The MLP was constructed with eight hidden layers with ReLU activations starting with a size of 1,024 neurons and halving in each subsequent hidden layer. A mean squared error (MSE) loss was used with an Adam optimizer and a learning rate of 0.001. It is worth noting that the MLP is very fast to train, taking only 170 seconds after features had been extracted. We refer to this combination of the unsupervised features and a regression model head for cell counts as a multi-output Cell Count Regression Network or CellCoRe.

Baseline Models and Performance Analysis

For comparison with CellCoRe, ResNet50 [75] and GoogLeNet [129] pre-trained on the ImageNet dataset were also used as baseline feature extractors in combination with the same MLP architecture.
To assess predictive performance, multiple metrics are used: Pearson’s correlation coefficient $r$, Spearman’s rank correlation coefficient $\rho$, and mean absolute error (MAE). For this problem, while a linear relationship between predicted and actual cell counts of different types of cells as measured by $r$ is desirable, a monotonic relationship quantified by $\rho$ is also useful as the output may be used for prioritizing or ranking slide regions. MAE is included to give an indication of the absolute difference between the predicted and actual values and for comparison with other methods.

### 4.1.4 Results & Discussion

We used the breast tissue images from the extended PanNuke dataset [84], [85] for the task of prediction of cellular composition. This dataset consists of three pre-defined folds. The dataset consists of 2,351 RGB H&E images of $256 \times 256$ pixels along with instance cell type maps. The cell type maps originally have six classes corresponding to cell types: neoplastic, inflammatory, connective/soft tissue, epithelial and dead cells and non-cell objects. In this analysis, the dead cells and background classes are excluded. Dead cells were excluded because for breast tissue, there are as few as four non-zero dead cell examples split across the three folds. Background non-cell objects were excluded because the distribution of labels is very different to the other classes. This leaves four classes: neoplastic, inflammatory, connective and epithelial. For each image, the corresponding cell segmentation mask and classification labels were transformed into a count vector of the cell types by counting.
the number of unique non-zero values in the channel of the instance mask corresponding to the cell type.

CellCoRe matches or outperforms both baseline models in every metric shown in Table 4.1.1 across all cell types. Furthermore, it also outperforms a segmentation-based method, DIST [130], in all cell types except for connective. This indicates that the unsupervised representation learning on domain data provided a better feature extraction than the generic ImageNet weights. It can also be seen in Table 4.1.1 that CellCoRe with an MLP regression head matches HoVerNet, the highest overall performing model, in Pearson’s correlation \( r \) for two of the cell types, inflammatory and epithelial cells. Overall it has a competitive \( r \) of 0.83, outperforming ResNet50 and GoogLeNet pre-trained on ImageNet. The linear relationship indicated by a high \( r \) can be seen clearly in the hexbin plots in Figure 4.1.5. It can also be seen that connective is the most noisy in the predictions.

The Spearman’s rank correlation \( \rho \) for CellCoRe and the baseline models are lower than their corresponding linear Person’s correlation
and are not as high as for HoverNet, and some cases DIST. This could be due to the segmentation models having access to significantly more informative pixel-level segmentation data in training. Additionally, there are many true cell counts grouped closely together in the range 0 – 5. Here, noise in the predicted cell count could greatly affect the rank order.

<table>
<thead>
<tr>
<th>Cell Type</th>
<th>Image 1</th>
<th>Image 2</th>
<th>Image 3</th>
<th>Image 4</th>
<th>Image 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neoplastic</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Inflammatory</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Connective</td>
<td>8</td>
<td>13</td>
<td>7</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Epithelial</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 4.1.6: Visualization of ground truth labels overlaid on sample image patches from the PanNuke-Breast dataset, with a corresponding table showing ground truth counts and predictions for cell count. Note that HoVerNet uses segmentation masks and classification labels for training, whereas our method does not use any such annotations or labels, requiring only the count of each cell type.

Connective cell counts appear to be the most difficult to predict, with all models performing the most poorly for this class. This could be due to the irregular shape of these cells which can have very long thin nuclei. In contrast, the other cell types are very circular.

It is interesting to note that Pearson’s correlation performance increases for both regression and segmentation models as the number of parameters increases. This is shown in Figure 4.1.7. The trend holds across many models, although it appears to plateau at an $r$ of 0.89.

While this self-supervised method shows promise for working
Figure 4.1.7: Scatter plot of parameter count versus Pearson's correlation $r$ for several models on the task of predicting the four-class cellular composition on PanNuke-Breast including Mask R-CNN [131] and Micro-Net [132].

Table 4.1.1: Comparison of test metrics for cell counts (Pearson’s correlation coefficient $r$, Spearman’s rank correlation coefficient $\rho$, mean absolute error (MAE)) between segmentation networks baseline networks (DIST and HoVerNet), baseline CNNs pre-trained on ImageNet (ResNet50 and GoogLeNet), and CellCoRe on the PanNuke dataset breast images. Values shown are the mean across test folds with the standard deviation in brackets. The highest two values for each row are shown in bold.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$r$</td>
<td>0.93 (0.02)</td>
<td>0.80 (0.06)</td>
<td>0.84 (0.01)</td>
<td>0.82 (0.04)</td>
<td>0.88 (0.01)</td>
</tr>
<tr>
<td>Neoplastic</td>
<td>$\rho$</td>
<td>0.89 (0.02)</td>
<td>0.79 (0.04)</td>
<td>0.78 (0.00)</td>
<td>0.77 (0.04)</td>
<td>0.86 (0.01)</td>
</tr>
<tr>
<td></td>
<td>MAE</td>
<td>2.33 (0.16)</td>
<td>3.46 (0.27)</td>
<td>3.70 (0.28)</td>
<td>3.85 (0.21)</td>
<td>3.16 (0.18)</td>
</tr>
<tr>
<td>Inflammatory</td>
<td>$r$</td>
<td>0.83 (0.01)</td>
<td>0.71 (0.10)</td>
<td>0.82 (0.03)</td>
<td>0.83 (0.04)</td>
<td>0.89 (0.01)</td>
</tr>
<tr>
<td></td>
<td>$\rho$</td>
<td>0.73 (0.00)</td>
<td>0.66 (0.01)</td>
<td>0.51 (0.01)</td>
<td>0.55 (0.02)</td>
<td>0.51 (0.03)</td>
</tr>
<tr>
<td></td>
<td>MAE</td>
<td>1.15 (0.07)</td>
<td>1.55 (0.27)</td>
<td>1.73 (0.06)</td>
<td>1.70 (0.09)</td>
<td>1.55 (0.09)</td>
</tr>
<tr>
<td>Connective</td>
<td>$r$</td>
<td>0.77 (0.01)</td>
<td>0.70 (0.01)</td>
<td>0.60 (0.04)</td>
<td>0.62 (0.01)</td>
<td>0.62 (0.02)</td>
</tr>
<tr>
<td></td>
<td>$\rho$</td>
<td>0.79 (0.02)</td>
<td>0.75 (0.01)</td>
<td>0.62 (0.03)</td>
<td>0.64 (0.01)</td>
<td>0.64 (0.01)</td>
</tr>
<tr>
<td></td>
<td>MAE</td>
<td>1.89 (0.05)</td>
<td>2.24 (0.10)</td>
<td>2.61 (0.07)</td>
<td>2.61 (0.06)</td>
<td>2.55 (0.03)</td>
</tr>
<tr>
<td>Epithelial</td>
<td>$r$</td>
<td>0.93 (0.02)</td>
<td>0.83 (0.05)</td>
<td>0.83 (0.01)</td>
<td>0.84 (0.03)</td>
<td>0.92 (0.01)</td>
</tr>
<tr>
<td></td>
<td>$\rho$</td>
<td>0.84 (0.05)</td>
<td>0.73 (0.02)</td>
<td>0.65 (0.01)</td>
<td>0.65 (0.01)</td>
<td>0.74 (0.02)</td>
</tr>
<tr>
<td></td>
<td>MAE</td>
<td>1.55 (0.25)</td>
<td>2.91 (0.40)</td>
<td>3.22 (0.18)</td>
<td>2.97 (0.22)</td>
<td>1.92 (0.15)</td>
</tr>
<tr>
<td>All Types</td>
<td>$r$</td>
<td>0.89 (0.02)</td>
<td>0.76 (0.05)</td>
<td>0.77 (0.02)</td>
<td>0.78 (0.03)</td>
<td>0.83 (0.01)</td>
</tr>
<tr>
<td>(Mean)</td>
<td>$\rho$</td>
<td>0.81 (0.01)</td>
<td>0.73 (0.01)</td>
<td>0.64 (0.01)</td>
<td>0.65 (0.01)</td>
<td>0.69 (0.00)</td>
</tr>
<tr>
<td></td>
<td>MAE</td>
<td>1.73 (0.07)</td>
<td>2.54 (0.24)</td>
<td>2.81 (0.02)</td>
<td>2.78 (0.09)</td>
<td>2.29 (0.11)</td>
</tr>
</tbody>
</table>

131
with patch-level regions of WSIs, it is not suitable for inference on larger patches alone. To incorporate a larger context, it could be used in combination with other methods such as simple majority voting, multiple instance learning methods, SlideGraph+ [62], or neural image compression [49]. Alternatively, a multiscale representation could be explored to capture features from multiple resolutions.

4.2 Wavelet Compressed Representations for Large Patches

WSIs are challenging to work with, primarily due to their extremely large size. WSIs obtained from commercial slide scanners are several orders of magnitude larger than natural images, such as those in the ImageNet [127] database, commonly used to benchmark artificial neural networks.

A typical brightfield WSI is a three-channel RGB image with spatial dimensions of 80,000 to 100,000 in width and height. Decoding the entire image uses a large amount of memory. For example, a WSI of 80,000 × 100,000 occupies eight gigapixels or 22 gigabytes of memory as eight-bit unsigned integers. As such, it is currently impractical to train a model using an entire WSI as an input sample due to limited GPU memory. Instead, many methods work on small image regions, often 224 × 224, also known as patches. This typically involves asking a pathologist to annotate regions of the WSI, followed by extracting patches from these regions to serve as training data. Common patch-wise prediction tasks include tissue (tex-
ture) classification, nucleus detection or segmentation, and gland segmentation. Independent patch predictions from these tasks can be aggregated and used as features for downstream statistical analysis. For example, detecting immune cells and tumour areas used to calculate a co-occurrence score of high prognostic value [133], [134].

However, a more compressed representation could be an alternative to the uncompressed three-channel RGB format often used for training CNNs. Using a compressed representation has the potential to allow for training on larger visual fields, giving the model context from a larger area. This could reduce or eliminate the need for aggregating features across patches.

Another challenge when working with histology images is their inherently multi-resolution nature. Salient histological features are apparent at different magnifications, for example, cell nuclei are around 4 µm (32 px at 40× magnification) across and large-scale glandular structures can be 1 mm (8,000 px at 40×) or more across. Pathologists work in a multi-resolution manner, starting at a low magnification to identify regions of interest before zooming into a higher magnification to examine finer details. This is also reflected by the image formats used for WSIs. They are typically tiled pyramid TIFFs or JP2 images that allow for fast access at different locations and apparent magnifications.

Although WSIs are encoded as a three-channel mapping of red, green, and blue light intensity, examination of stained pathology samples is concerned with the intensity of histological stains such as haematoxylin and eosin rather than red, green, or blue. As such it
is possible to perform a stain separation transformation via convolution with a stain matrix and produce a new three-channel image. For the most common case of H&E staining, the first two channels correspond to the concentration of haematoxylin and eosin respectively. The third channel may be used to represent another stain or as a residual channel. If the third channel is irrelevant to the task, it can be removed to reduce the data volume by one-third. Even with this data reduction from dropping a channel, the data is often still too large to feed forward through a neural network on a GPU other than via patch-based methods.

Some methods have been shown to train well on JPEG compressed images by partially decompressing the JPEG encoding into a block-wise discrete cosine transform (DCT) domain representation and redefining CNN operations such as 2D convolution and providing approximations to non-linearities such as the ReLU activation function [135]. The main drawback is that this requires re-implementing most if not all operations in machine learning libraries such as TensorFlow and PyTorch for a model such as ResNet. There is currently no support in existing distributions of these libraries for these redefined operations that can be applied to a JPEG DCT domain data representation. Additionally, the existing operations have already been highly optimize for regular image representations.

Some existing methods have been proposed to enable the training of neural networks on large histology images in order to solve problems which require or benefit from a larger context [49], [53]. However, these methods depend on first learning a local representa-
Figure 4.2.1: A patch-based method A versus the proposed B which extracts a large region from a WSI and applies a transform which reshapes and reduces the volume of data.

tion which is not updated by the second stage model which learns the larger context. Not gradient flow occurs from the larger context model to the local patch-level network.

This work proposes an alternative approach via the application of a space-frequency transform to the RGB or stain channels separately. The generated sparse frequency response maps are resampled to create a multichannel image with the channels encoding information from different orientations and frequencies corresponding to different magnifications. This reduces the image to a feature cube with an overall lower volume of data (Figure 4.2.1). This new multichannel spatial frequency image can then be used as input to common CNN architectures such as ResNet [75].

The results show that a significant reduction in the input data
size can be achieved via spatial frequency transforms such as the DCT and DWT in addition to pooling or coefficient culling without a significant impact on the accuracy or training time. Additionally, larger input patches may be used with this method to incorporate more context and multi-resolution features.

4.2.1 Methods

Initially, an image on disk is decoded into an \(m \times n \times 3\) RGB array. Optionally, colour space conversion can be applied at this stage for example to YCrCb, CIELAB, or to stain optical density. Next, a spatial frequency transform is applied, channel-wise, to the \(m \times n \times c\) input image with \(c\) channels. Here two methods of spatial frequency were tested: the discrete cosine transform (DCT) and the discrete wavelet transform (DWT). A data reduction step is then applied to the resulting coefficients by either dropping coefficients or applying a pooling operation. Lastly, the resulting values from each channel are stacked to form a three-dimensional channel and each of these is concatenated to form the final multichannel image which can be used as input to a conventional neural network such as AlexNet [42] or ResNet [75] with small modifications to the input layer to accommodate the new input shape.

Experiments were run for two combinations of spatial frequency transform and data reduction: a DCT with coefficient culling by removing high-frequency coefficients and a per-channel multi-level 2D DWT with max pooling based on the absolute magnitude of the sub-band coefficients.
Block-Wise DCT with Coefficient Culling

A block-wise type-II discrete cosine transform (DCT) is applied to the spatial image array with a block size $b \times b$ pixels. This is similar to the JPEG encoding algorithm, which applied the DCT over fixed size $8 \times 8$ pixel blocks. A subset of $c$ coefficients from each channel is selected in a diagonal ordering from the 0th basis function, which is effectively an average value for the block. The selected coefficients are stacked into a multichannel image of shape $\frac{m}{b} \times \frac{n}{b} \times c$. This has the effect of producing a down-sampled image whilst also including additional frequency information for each pixel in the additional channels.

Multilevel 2D DWT with Max Pooling

Here the DWT is applied in two dimensions to each channel separately after the channel mean has been subtracted to centre the data around zero. This is beneficial for both the DWT and the neural network. The spatial image is decomposed for $k$ levels by repeated application of the discrete wavelet transform using some wavelet $w$. The produced set of sub-bands decreases in size by half for each level in the range 1 to $k$. Sub-bands of a higher spatial resolution than the last low frequency pass $LL_k$, which is effectively a down-sampled version of the input image, are pooled to produce a set of pooled frequency response maps which have a homogeneous resolution in width and height (Figure 4.2.2). This allows the sub-bands to be stacked into a $m \times n \times (l \times 3 + 1)$ multichannel image.
Figure 4.2.2: Diagram showing the steps of a two-level DWT with pooling for a single channel. A) Application of the DWT, B) A pooling operation, C) Stacking into a multichannel image. The pooling step shows the reduction in the data volume by the white area left in the dashed square.
Choice of Pooling Function

A max-pooling based on the absolute magnitude was chosen because the high-frequency sub-bands are sparsely populated with many of the coefficients near zero and contribute perceptually very little to the input image, a property exploited by JPEG 2000 compression. A max-pooling method prevents the magnitudes from being diluted by the rest of the pool. Additionally, since the coefficients are centred around zero, an average pooling could allow for many low-magnitude positive coefficients and a single high-magnitude negative coefficient to sum to near zero and lose information about significant changes in the intensity of the original image.
Choice of Wavelet

The LeGall-Tabatabai (LGT) 5/3, the same wavelet used for JPEG 2000 reversible was chosen. The LGT 5/3 lossless wavelet transform is the next simplest lossless wavelet transformation, following the Haar wavelet transformation. It has several desirable properties for transforming images, being the shortest symmetric biorthogonal wavelet. Being symmetric and biorthogonal is desirable because all orthogonal wavelets (other than the Haar wavelet) are non-symmetric and produce noticeable phase distortions in images. Being symmetric also helps to avoid boundary effects. As a short wavelet, the number of operations is also reduced, keeping the transformation fast. LGT 5/3 also has the possibility of being implemented using only integer coefficients. This has the potential for future optimization by performing an integer lifting scheme on data in place, which should be significantly faster than the simpler floating-point convolution-based method used in this work. Additionally, should this method show promise, it may be possible to use coefficients directly from JPEG 2000 files without fully decoding the image.

4.2.2 Experimental Setup

Two experiments were considered. Tissue classification was selected to show that the proposed wavelet method is non-inferior in tasks

---

1This numbering system is specifying the filter sizes. In this case, having 5 high pass filter coefficients and 3 low pass filter coefficients. An alternative numbering system for wavelets is via the number of vanishing moments by which the LGT 5/3 wavelet could also be known as a ‘biorthogonal 2.2’ wavelet.
that do not require a larger context, as this is a task that patch-based methods work already well for. Additionally, a colorectal adenocarcinoma grading task was chosen to demonstrate the performance on tasks which require more context from spatially related tissues and larger structural patterns.

PyTorch version 1.3.1 and torchvision 0.4.2 were used for training in all the experiments. The experiments were run using one Nvidia GTX 2080Ti, 64 GB of computer memory and an Intel Core i5-8500 CPU. The 2D discrete wavelet transform was implemented using a sequence of filter bank convolutions in PyTorch so that it could be applied quickly on the GPU to a batch of images after exiting augmentations had been applied. The DCT experiments used the transform from OpenCV.

**Tissue Classification (Kather Dataset)**

Tissue classification is a very common multi-class classification problem in computational pathology with patch-based methods commonly achieving a validation accuracy > 95%, typically with between 2 and 10 classes. The dataset used for this task was the NCT-CRC-HE-100K dataset\(^2\) of 100,000 patches produced by Kather et al. [94] from colorectal slides in the National Center for Tumour Diseases (NCT) biobank database. The slides which the image patches came from were all formalin-fixed paraffin-embedded (FFPE) H&E-stained slides. Each patch is labelled with one of nine classes and shows a small patch of tissue at an apparent 20× magnification,

\(^2\)The NCT-CRC-HE-100K dataset is available for download at [https://zenodo.org/record/1214456](https://zenodo.org/record/1214456)
approximately 0.25 microns per pixel with a resolution of $224 \times 224$ RGB pixels. The image format used for the patches is an uncompressed TIFF.

The AlexNet model was used for each run with a different input representation and was trained for 40 epochs on each run. The stride of the input layer was adjusted to the input dimension across runs. A learning rate of $1 \times 10^{-3}$ was used and a batch size of 160. Unlike in the work by Kather et al. [94], no pre-trained weights from ImageNet were used. Instead, random initialization was used. This was to try to ensure a fairer comparison between methods so that they would not be influenced by any loss in predictive power which may come from adapting pre-trained weights to the adjusted layer.

This dataset is of high quality and pre-prepared into image patches with no visual artefacts and high-quality staining. No preparation was required other than to produce an internal training and testing split. Internal subsets containing 70%/15% of the original dataset were used for training and testing in the same way that Kather et al. [94] did in order to have a more direct comparison of results. Additionally, the evaluation set of 7180 image patches was used as external validation. After training on the 70% subset, the model was retrained with the whole dataset and inference was performed on the external validation dataset.
Colorectal Adenocarcinoma Grading (Extended CRC Dataset)

Cancer grading was selected as a second task to evaluate the method because it requires incorporating information about both small-scale cell-level details and large-scale glandular structures. The size reduction of input data from this method enables training using larger patches which would otherwise not be possible due to GPU memory constraints.

The extended colorectal cancer grading dataset \(^3\) and a set of 3 folds produced by Shaban et al. [53] was used for this task. This dataset consists of visual fields obtained at 20× magnification and an average size of 7520 × 4548 (width × height) pixels extracted from over 100 WSIs. These have been labelled by a pathologist as normal, low grade, and high grade.

The ResNet18 neural network was trained using randomly cropped regions of 4,096 × 4,096 pixels within each larger visual field in the dataset. This was performed during training due to the varying size of the visual fields in the dataset. The smallest visual field height was 4,480 pixels. Therefore, a square crop of 4,096 was chosen as it would fit within all the visual fields.

The network was trained for 100 epochs with a learning rate of 1 × 10\(^{-3}\) and a batch size of 10. Random horizontal and vertical flipping were used to augment the training data and stain separation was applied. Additionally, for comparison, ResNet18 was trained with regions downsampled with bilinear interpolation also using random

\(^3\)The extended CRC dataset is available for download at https://warwick.ac.uk/fac/sci/dcs/research/tia/data/extended_crc_grading/
horizontal and vertical flips with stain separation.

For evaluation, a single centre-cropped region of each visual field was taken for inference. The label for this was used as the predicted class for the whole visual field.

This dataset is of high quality, with each image containing mostly tissue and little to no visual artefacts. As a result, no data perpetration was required beyond using the same internal training and test split from previous work by Shaban et al [53].

**Lung Carcinoma Subtyping (CPM Dataset)**

Two tasks, one of binary WSI classification and the other of WSI nuclei segmentation, were given as part of the Digital Pathology challenge during the MICCAI 2017 conference. Here this pooled wavelet method is applied to the WSI binary classification task. More precisely this task is to classify FFPE H&E WSIs containing tissue from one of the two road classes of lung cancer called non-small-cell lung carcinoma (NSCLC), as opposed to small-cell lung carcinoma (SCLC), into the subtypes NSCLC adenocarcinoma (LUAD) or NSCLC squamous cell (LUSC).

For this dataset, in contrast with the tissue-dense visual fields in the CRC task, the WSIs often contain quite a large proportion of background which is of no diagnostic value. Additionally, several slides contain multiple serial sections, meaning slices taken at varying depths within the same tissue block. Lastly, the WSIs were encoded using a vendor-specific compressed TIFF variant which was slow to decode and load into memory compared with other WSI for-
First, a thumbnail of the slide was generated using the OpenSlide [64]. Otsu’s threshold method [30] was applied to the thumbnail followed by morphological operations to remove very small regions which could occur due to dust or scratches on the slide. Finally, the minimum bounding rotated rectangle of each connected component was identified. Subsequently, each of these tissue regions was extracted to an uncompressed TIFF file and assigned the slide-level label. For the majority of slides, only a single region filling most of the slide was identified. However, in the case of serial sections, multiple regions were extracted for each slice of tissue in the slide. This eliminated the processing of many background pixels. It also functioned as a natural data augmentation, as the serial sections are very similar in appearance but not completely identical.

The dataset was divided into training and testing subsets using the same split as from the previous patch-based method by Vu et al. [136]. The internal training and test sets contained 32 cases each with an even 50% split between LUAD and LUSC.

4.2.3 Results

Results are given using the same training, testing, and evaluation data splits as other methods and are compared via the same performance metrics of accuracy and weighted accuracy.
Tissue Classification (Kather Dataset)

When comparing runs which use a similar percentage of the original data volume, the wavelet-based methods perform better with 100% and approximately 15% of the original volume on both the internal and external test sets. At around 44%, the bilinear downsampling appears to slightly outperform the wavelet method on the external test set. This could, however, be due to variance in model convergence.

The block-wise DCT input representation does not perform well on the internal or external test set, except for when compared to the 90x90x3 downsampled representation which performed particularly poorly.

In comparison, Kather et al. [94] trained several common networks: VGG19 [137], AlexNet [42], Squeezenet 1.1 [138], GoogLeNet [129], ResNet50 [75]. They achieved the highest accuracy of 98.7% on the internal validation set with VGG19. AlexNet via transfer learning from ImageNet achieved an accuracy of approximately 97.1% (exact numbers were not given) and a training time of approximately one hour.

There is an increase in training time from using the DWT on the inputs after other augmentations such as vertical and horizontal flips have been applied. The 2-level DWT with max pooling adds an overhead of approximately 396 µs per image patch processed to the training time versus the original. This is comparable with other simple augmentations such as colour space shifts, Gaussian
blurs, and edge filters in popular augmentation or image processing libraries can take orders of magnitude longer to run on the CPU [74].

**Colorectal Adenocarcinoma Grading (Extended CRC Dataset)**

The accuracy of the multi-level DWT and max pooling compares favourably with the patch-based method used by Shaban et al. [53] (Table 4.2.2) where the ResNet50 and Xception [139] models were trained on small glandular patches to predict one of the three grade labels or a background label which was added by masks generated from a separate gland segmentation network with refinements done by hand. Additionally, it performs well in comparison with the context-aware model proposed by Shaban et al. which was shown to outperform other methods.

Accuracy from Shaban et al. [53] is averaged across the folds with the variance across those given. The accuracy for the DWT and bilinear downsampling is given for the mean and standard deviation of the last ten epochs of the train and test split of the first fold.
Table 4.2.1: Input representations and best internal validation accuracy for AlexNet trained on 70% of the NCT-CRC-HE-100K dataset for 40 epochs and best external validation accuracy. The accuracy train time is the time taken to train on the internal set of 70,000 patches.
<table>
<thead>
<tr>
<th>Input Representation</th>
<th>Shape</th>
<th>Values</th>
<th>Accuracy % (3 Class)</th>
<th>Weighted Accuracy % (3 Class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50 (Shaban et al.)</td>
<td>$224 \times 224 \times 3$</td>
<td>150,528</td>
<td>86.33 ± 0.94</td>
<td>80.56 ± 1.04</td>
</tr>
<tr>
<td>Xception (Shaban et al.)</td>
<td>$448 \times 448 \times 3$</td>
<td>602,112</td>
<td>86.67 ± 0.94</td>
<td>80.42 ± 1.25</td>
</tr>
<tr>
<td>Context-Aware (Shaban et al.)</td>
<td>$1792 \times 1792 \times 3$</td>
<td>9,633,792</td>
<td>86.67 ± 1.70</td>
<td>84.17 ± 2.36</td>
</tr>
<tr>
<td>Bilinear Downsample (ResNet18)</td>
<td>$512 \times 512 \times 3$</td>
<td>786,432</td>
<td>51.71 ± 8.18</td>
<td>55.85 ± 25.95</td>
</tr>
<tr>
<td>DWT 3 Level Max Pool (ResNet18)</td>
<td>$512 \times 512 \times 30$</td>
<td>23,520</td>
<td><strong>88.33 ± 3.62</strong></td>
<td><strong>90.50 ± 4.79</strong></td>
</tr>
</tbody>
</table>

Table 4.2.2: Input Representations with accuracy and weighted accuracy for colorectal adenocarcinoma grading.

**Lung Carcinoma Subtyping (CPM Dataset)**

Results in line with previous work by Vu et al. [136] can be seen in Table 4.2.3 with the pooled wavelet approach obtaining the same best accuracy using the same train and test data split.

**Visualization**

To gain a better understanding of the salient features in the images in which the network is influenced by a visualization method, Score-CAM [140] was implemented to produce activation maps for each class (Figures 4.2.4 and 4.2.5). Score-CAM is an easy-to-implement visualization method which does not rely on network gradients. In addition to having a simple implementation, Wang et al. propose that by not relying on gradients, Score-CAM is less susceptible to issues such as gradient noise, vanishing gradients, and false confidence. They also demonstrate that Score-CAM activation maps are more densely concentrated on the subject in a photograph.
Table 4.2.3: Lung subtyping (CPM dataset) results showing the best accuracy. For the patch-based methods by Vu et al. using ResNet32, MV is max voting aggregation and RF is a random forest classifier aggregation.

The resulting Score-CAM activations maps were also examined in detail using an interactive web user interface, described later in section 5.3.2 (Aura: Heatmap Visualization).

As seen in Figures 4.2.4 and 4.2.5, activation is concentrated on glandular regions. This is consistent with histological guidance which specifies glandular differentiation as a diagnostic pattern for grading colorectal adenocarcinomas [90].

4.3 Chapter Summary

This chapter examines two alternative histology image representations: self-supervised and a pooled multi-level wavelet (DWT) transform.

The self-supervised method demonstrates that pre-training a popular CNN architecture with a representation learning task and unlabelled domain data can produce embeddings that map well to labelled data, has the potential to detect erroneously labelled data, and can produce competitive cellular composition predictions. The proposed framework could be further improved by introducing more
Figure 4.2.4: Score-CAM activation map visualization of class activations for a correctly predicted low-grade colorectal adenocarcinoma visual field.

Figure 4.2.5: Score-CAM activation map visualization of class activations for a correctly predicted high-grade colorectal adenocarcinoma visual field.
extensive and domain-specific augmentation in the training data. This paradigm could be applied to other tissue types with retraining of the regression head, and the features from the SwAV representation learning could be utilized for multiple tasks simultaneously.

The compressed wavelet representations utilize the DWT with max pooling to reduce data size without negatively affecting accuracy. This method allows for larger input visual fields and may reduce or eliminate the need for annotation or aggregation and context learning across patches. The DWT method could be extended to other clinical problems and shows potential for improving predictions.
Chapter 5

Annotating Histology Images

In the context of digital and computational pathology, annotation is the labelling of features in a digital pathology image, such as an extracted visual field or an entire WSI. Annotation can be done at different levels of granularity, from simply labelling the entire slide as ‘normal’ or ‘abnormal’, outlining large structures and regions of tissue, drawing precise boundaries around individual cells, or even highly detailed pixel-level annotation.

Annotation data is of critical importance to training models. It is used as the ‘ground truth’ for the model to compute gradients during supervised training. It is also crucial for validating the performance of both supervised and unsupervised methods, for example, by computing metrics such as the F1 score or area under the receiver operating characteristic (AUC-ROC).

However, there are many challenges associated with gathering
annotation data. One big challenge is that it can be incredibly time-consuming and labour-intensive. Pathologists must carefully examine each image and label all the relevant features. Another challenge is that obtaining consensus annotations from multiple pathologists can be difficult. Different pathologists may have different interpretations of the same image.

It can also take much work to ensure the quality of annotation data. How to draw an annotation and where the boundary between regions lies can be subjective. The pathologists’ experience and biases influence the quality and detail of annotation. Obtaining enough consistently high-quality annotation data is often very challenging.

Furthermore, annotations may also be generated as the output of a model or algorithm, such as cell boundaries from a HoVerNet model or segmented tissue regions from a U-Net model. For an automated model, output there may be an enormous number of annotations to handle. For example, a WSI may have millions of detected annotated cell boundaries. This dataset’s storage and querying can become quite challenging. These annotations may occupy significant disk space, and it is often infeasible to keep many annotations in the memory of a desktop workstation. Furthermore, searching for relevant annotations when performing downstream analysis may be slow if a naïve methodical scanning method is used. Additionally, visualizing the output of automated annotation can be difficult due to the WSI size and the large number of objects to display.

As the field of pathology continues to evolve, annotation is likely
to remain vital as a tool for training machine-learning models and improving the speed or accuracy of diagnosis. Even with unsupervised methods showing great potential, pipelines must undergo extensive validation before being used in a clinical setting.

5.1 Representing & Storing Annotations

In TIAToolbox, we refer to the combination of a geometric entity and its associated properties as an annotation. A geometric entity may be a point, polygon, sequence of line segments (line string) or a closed line string loop with no area (linear ring). Properties are defined to be a hierarchical JSON-like structure which may contain strings, integers, floats, dictionaries and lists.

In implementing the SQLiteStore for TIAToolbox, annotation geometry is kept in a zlib compressed well-known binary (WKB) format. Zlib compression reduces the space required to store geometries but at the cost of some additional encoding and decoding time. Point annotations are an exception, where no WKB is stored for them. An R*-Tree index is included in the database for all annotations to accelerate spatial queries (see section 5.2 for more on querying). Each R*-Tree index row corresponding to a point annotation contains all the necessary information to recreate the point. Therefore, no additional storage is used for storing points as WKB.

However, this was just one of many options that were explored before arriving at this solution. Before deciding on the final format, other representations that were considered included:
• Serialized NumPy Arrays

• Ragged arrays

• GeoJSON, as used in Salsa and Tango.

• New line delimited GeoJSON (NDJSON)

• Well-known binary (WKB)

• Well-known text (WKT)

Initially, storage of point annotation was considered by simply serializing the $n \times 2$ array of numeric values in C array ordering using NumPy’s simple binary .npy format. This format is very space efficient and fast to read and write. However, for large point sets, spatially querying this data format was slow, as the whole set of points but be scanned through each time. Lastly, this format is limited to point annotation data only. It does not accommodate storing other variable-length geometric types such as polygons or associated data, which may also be unstructured or of variable length.

As implemented by libraries such as Awkward Array, ragged arrays can handle variable-length data such as polygons. However, this also needs to solve the issue of queries needing to be faster at scale. Furthermore, this only defines storing and processing the data in memory, relying on other formats such as JSON, Arrow, or Parquet for serialization on disk.

GeoJSON is a simple and popular JSON-based format for representing geometric data with arbitrary additional properties. It also has the benefit of being both human and machine-readable, which
can assist with testing and debugging. However, it is a more verbose and space-inefficient format than other options. Furthermore, for large GeoJSON files, parsing the serialized data with standard JSON parsing libraries may consume a large amount of memory, potentially more than the machine can access. This is because the entire file must be loaded into memory for parsing. There are some implementations of JSON parsing which can process data in a more memory-efficient way. However, these are typically limited to certain situations or require additional information before parsing, such as the length of an array.

To address this memory consumption issue, NDJSON was explored. New line delimited JSON, nd-json or NDJSON, is a file format where each line of the file is a self-contained valid JSON document. This format enables the file to be parsed in chunks, line by line. NDJSON is ideally suited to large annotation datasets where each line may be a separate annotation as a GeoJSON document. However, it is still a verbose and space-inefficient format. As the structure of GeoJSON has much redundancy with many frequently reoccurring patterns and symbols. However, this predictable structure suits LZW, DEFLATE, ZStandard, and Gzip compression well. Furthermore, many compressors such as Gzip and ZStandard may be used via a streaming API so that chunks of data can be parsed whenever a new line character sequence is encountered. New-line delimited GeoJSON (nd-GeoJSON) has excellent potential as an archival format for annotation, especially when compressed. This is due to its simple structure and ability to be read with standard
off the shelf tools. For example, it may be easily read and written using only standard library modules in Python.

Lastly, well-known binary (WKB) and text (WKT) formats are well-defined and commonly used to represent geometries in geographic information systems. These were explored as a compact alternative to the more verbose GeoJSON. They are also more compact representations of geometry than GeoJSON, with WKB slightly more compact than WKT in exploratory testing. Furthermore, WKB compressed well to reduce further the space required.

Finally, SQLite was chosen as a container to organize a collection of WKB binary blobs and enable fast querying. SQLite offers in-memory processing and single-file format with comprehensive platform support and rich features including support for spatial indexing via an R*-tree index, JSON querying, custom indexes, and much more. SQLite does have an optional module for directly supporting a binary form of GeoJSON. However, this not enable by default on standard builds of SQLite. When writing annotation storage classes in TIAToolbox, the builds of SQLite included within distributions of Python for Ubuntu, MacOS, and Windows came with enabled support for R*-tree indexes but not for GeoJSON data. For this reason, it was decided not to rely on this optional feature. Lastly, one significant advantage of using SQLite for annotation storage is that it can be accessed concurrently across multiple threads or processes.
Compressing WKB

In addition to exploring several serialization formats, a range of compression methods were also explored. This comparison was performed by generating, serializing, and compressing 1,000 polygons representing artificial cell boundaries. Polygons were generated as a closed oval of twenty vertexes with randomized eccentricity, orientation, and added noise. Each polygon also had a random offset in X and Y between 0 and 100,000. After serializing to WKB, several compression methods were applied:

- Blosc meta-compressor using Zstandard
- Blosc2 meta-compressor using Zstandard
- Brotli with compression level 9
- Brotli with compression level 11
- DEFLATE
- PackBits
- LZ4
- Zlib
- Zlib with the signature magic number removed
- Zstandard (zstd)
- Zstandard (zstd) with the signature magic number removed

Figure 5.1.1 compares the time it takes to compress data and the size of the compressed data relative to using no compression. Here
is a trade-off between the compression time and the data size, which follows a power law. We chose to use Zlib because it balances size and compression speed well. Additionally, Zlib is already included in Python’s standard library, which avoids additional dependencies. In TIAToolbox, a user can choose to compress data or not. This is specified in the metadata table in the SQLite database. This also allows for other compression methods to be used in the future.

5.1.1 Implementation in TIAToolbox

TIAToolbox implements two storage classes. One is DictionaryStore, which is a class backed by a simple in-memory dictionary (hash table) Python object and is well-suited for small annotation sets and single-process applications. It also serves as a simple in-memory baseline implementation for annotation storage. At the same time, SQLiteStore uses the proposed SQLite zlib compressed well-known
binary (WKB) alongside additional hierarchical properties stored as JSON. Example usage of the SQLiteStore class is shown in Appendix A.10.

Both of these stores are based on an abstract base class that defines an interface. This means that we can easily add implementations that use different back-ends, like a geospatial database, in the future. Additionally, the base class extends Python’s MutableMapping interface to make the storage classes feel more ‘Pythonic.’ Subclasses also get access to functions that do not rely on the specifics of the back-end store or can be created from other base class functions. For instance, the base class has functions for converting to and from other formats that are available to all subclasses. A store can be created from or exported to several formats, including a Pandas DataFrame, GeoJSON feature collection, Line-delimited JSON, NDJSON, or a Python dictionary.

5.2 Querying for Annotations

The SQLite store has been optimized to handle millions of annotations. To achieve this, an R*-Tree index is used for fast spatial queries based on bounding boxes. This index is a data structure that reduces the search space for the query by performing a bounding box intersection. Instead of searching through all annotations, the R*-Tree index filters only those within the specified bounding box. This optimization makes it possible to query large datasets with millions of annotations much faster and more efficiently. Additionally, the
bounding box query is extended to include binary shape predicate testing through a custom function call-back to Python. This second filtering stage ensures that queries return only annotations that intersect with or are covered by a geometry provided to the query function.

To refine the search for annotations, constraints can be specified on JSON properties along with querying for annotations that intersect a given geometry. We utilize a restricted subset of the Python language to provide a simple domain-specific language (DSL) and thus enabling predicate statements to be supplied to queries and evaluated efficiently for a specific backend where possible. SQLite offers expressions to filter results based on JSON within a table column during a query, which provides improved performance over Python interpretation which is notoriously slow and restricted by the global interpreter lock (GIL). However, certain operations may not be feasible through SQLite’s variant of structured query language (SQL), and some backends may not support JSON querying. This DSL can be mapped to an optimized backend implementation wherever possible, and if not, it can fall back to post-filtering within the Python interpreter, providing a fallback to simple post-query filtering in Python should there be no optimization available.

5.2.1 Performance & Accelerating Queries

A benchmarking notebook is included in the toolbox source repository to demonstrate the performance benefits of our SQLite-backed storage class [141]. This notebook performs several common tasks
Figure 5.2.1: A Time in seconds to append 1,000 annotation for Dictionary Store and SQLiteStore. B Time to append 1,000 annotation and serialise to disk. DictionaryStore uses line delimited geoJSON here for serialization. The dashed line shows 100 ms as a point of reference. The bar height indicates the minimum run time and the error bar extends to the maximum run time across the 1,000 append operations.

on generated grids of cell boundary polygons. This notebook shows that appending geometry is more expensive for SQLiteStore. This is because the indexes must be updated on insert. However, querying after insertion and indexing is much faster. For example, performing a simple box query is approximately three times faster with the SQLite store for a 100 × 100 (10,000) annotation grid (Figure 5.2.1). For more complex queries, the gap in performance increases. SQLiteStore can be over one hundred times faster when performing a polygon intersection query (Figure 5.2.2). However, benchmark times vary depending on hardware\(^1\). In benchmarking on a spatial box query over a set of five million point annotations, SQLiteStore performed over 48,000 times faster than a full-scan through a dictionary store and 56 times faster than a NumPy array (Figure 5.2.3 A).

\(^1\)Results shown here are from running on Google Colab with an Intel(R) Xeon(R) CPU running at 2.20GHz and 12 GiB of RAM.
Figure 5.2.2: Bar plot showing the time for 100 polygon intersection queries using DictionaryStore and SQLiteStore. **A** is without an additional predicate filter on JSON properties, and **B** is with a predicate. The dashed line shows 100 ms as a point of reference. The bar height indicates the minimum run time and the error bar extends to the maximum run time across the 100 append operations.

Figure 5.2.3: **A** Time to query five million point annotations with DictionaryStore, SQLiteStore and as a NumPy array. **B** Time to query using a predicate with and without an index. The dashed line shows 100 ms as a point of reference. The bar height indicates the minimum run time and the error bar extends to the maximum run time across the 100 append operations.
The **SQLiteStore** class also comes equipped with a convenient utility function that creates an index based on expressions written in the DSL. Indexes can enhance the speed of common queries. SQLite’s query planner algorithm can vary in complexity, but for non-indexed queries, a simple full-table ‘scan’ method is utilized [142], which visits every row in $O(n)$ time. Using an index can reduce the complexity to $O(\log n)$ for value lookups. This is due to the use of binary search over the sorted values in the index. By utilizing a JSON column and indexes, users can take advantage of the flexibility and convenience of a schema-free JSON hierarchical structure while also having the option to use indexes to expedite common value lookups.

There are various convenient query methods available to optimize annotation analysis. Two examples are the property only query (\texttt{pquery}) and bounding box query (\texttt{bquery}) methods. The property only query method allows for searching property values without the need to fetch, decompress, or parse geometry data. Combining values into sets while processing results to save memory is also possible. This method is particularly useful for checking unique values. Visualization tools, such as the one used in the IGUANA [143], [144] paper, utilize this method to identify unique property values in the dataset efficiently.

The bounding box query (\texttt{bquery}) is a useful method for performing spatial queries that need to know the approximate area and location of annotations but not full geometry detail. This method performs a bounding box spatial query using the R*-tree index but
avoids fetching, decompressing, and parsing the full geometry. Only the bounding box of the annotation is returned. An example use case is for visualizing a whole dataset as a series of bonding boxes at a low resolution, for example, to generate a thumbnail representative of the annotation dataset.

**Neighbourhood Queries**

Looking for annotations within proximity to other annotations can be a helpful task. In other words, identifying neighbourhoods of annotations. For example, it is important to locate immune cells, such as tumour-infiltrating lymphocytes (TILs), within a specific radius of tumour cells to predict disease progression accurately. TILs are an example of ‘anti-tumour immunity’ and are a critical prognostic indicator [145], [146]. Therefore, precise identification of these cohabiting cells is crucial.

Another use for query neighbouring annotations is to identify duplicate or overlapping annotations. This is a crucial step in post-processing for algorithms that process a whole slide image in overlapping patches. The Hungarian algorithm ² is one method which can be used for reconciling results across overlapping patches. This algorithm solves the assignment problem, finding the minimum cost assignment of ‘agents’ to ‘tasks’, and is used in some algorithm implementations, such as HoVerNet. However, the algorithm can be quite slow in practice. The original version has a time complexity of $\mathcal{O}(n^4)$ [147], and other algorithms can solve the problem with a

\[\text{Also known as the Kuhn-Munkres algorithm or Munkres assignment algorithm.}\]
time complexity of $O(n^3)$ [147]. However, duplicate and overlapping annotations may also be identified by pairwise comparison with a limited set of neighbouring annotations. This is a relaxed problem and can be solved with lower time complexity.

When looking for annotations that satisfy two distinct filter functions and are within a specific distance of one another, a straightforward implementation with nested loops can be utilized. However, this approach has a time complexity of $O(\epsilon)$. To decrease the number of annotations that require checking by the inner loop, the spatial index in SQLiteStore can be utilized. This reduces the average complexity of the inner loop to $O(\log n)$. By employing this technique, the overall time complexity of a neighbourhood query utilizing SQLiteStore.nquery() is decreased to $O(n \log n)$.

The complexity could potentially be further reduced by using an index on the property for the initial predicate filter. Querying an index has average complexity $O(\log n)$. This would result in a superior overall complexity of $O(\log^2 n)$.

After its initial release, a method for neighbourhood querying was developed for TIA Toolbox. The method efficiently queries annotations within a given neighbourhood and is implemented in the base class from which DictionaryStore and SQLiteStore classes inherit.
Figure 5.2.4: Time in seconds to find overlapping polygon annotations between two copies of an $n \times n$ grid overlaid on top of each other. The left plot shows a linear scale and the right plot shows a log scale. Times for two implementations of the `nquery()` method are shown. These are for the `DictionaryStore` and `SQLiteStore` classes in TIA Toolbox. `DictionaryStore` uses a simple nested loop structure whereas `SQLiteStore` uses an R*-tree index to accelerate queries.

Figure 5.2.5: Line plot of time versus $n$ for `SQLiteStore.nquery` from $n = 10$ to $n = 120$ with reference plots of $y = n$, $y = n \log n$, and $y = n^2$ are shown for reference.
5.3 Web-Based Tools for Annotation & Visualization

Gathering annotations and collaborating between researchers and NHS pathologists can be challenging. For example, NHS workstations cannot install additional software which prevents the use of local desktop annotation tools. However, as workstations are equipped with a modern web browser, a web-based annotation application could allow for remote gathering of annotation and collaboration. As histology annotation requires specialized domain knowledge, unlike many natural image annotations (for example labelling an image as of a cat or dog), it is important to be able to reach experts who may be working remotely. Furthermore, such tools could be used locally by researchers for viewing and editing annotations.

5.3.1 Salsa: Web-Based Collaborative Annotation

Displaying and annotating WSIs on the web has many similarities to working with satellite imagery or a geographical information system (GIS). For many years, mapping services such as Google Maps have displayed information in web browsers using a ‘slippery’ map. This is a map made up of many tiles at multiple zoom levels, much like the pyramid of tiled images in a WSI file. This map can be panned and zoomed by the user clicking, dragging, and scrolling the mouse. Only the required tiles for the current region are sent from the server to the client. A slippery map is also well suited to displaying WSIs and annotation data. Existing client-side JavaScript slippery map
implementations can work with little or no modification to render WSI tiles received from a simple custom tile server.

There are many options for representing annotation data including pixel maps, XML, and more. One such representation is a popular JSON-based standard for representing geographical features, GeoJSON. This format is not only suitable for geographic features but geometric primitives such as points, lines, and polygons in general. As such, a good format for representing annotations on a WSI and is supported by some desktop digital pathology software, such as QuPath [65], as an annotation format.

A prototype application called Salsa was developed using the Leaflet JavaScript slippery map library Leaflet to display tiles and GeoJSON as the annotation format. The application utilized a
Django server to manage WSIs and serve tiles with a client-side JavaScript web application. WebSocket connections are utilized by both the client and server to maintain a persistent asynchronous connection. Additionally, a message-passing system allows for live collaborative annotation of WSIs over the socket connection. This system propagates edits to all connected users. It also transmits each user’s current location within the WSI so that others can ‘follow along’. This is particularly useful when talking through annotations or a slide over a video conference. To create a frame of reference in addition to a visible mouse on video conference screen sharing, an overlay grid is supported so that regions can be referred to by either absolute coordinates within the slide or relative screen coordinates. For example, the host may reference square ‘E7’ as others follow their view of the WSI and associated annotations.

Salsa also applies grouping of points into clusters via a Leaflet plugin (Figure 5.3.2). This is used when zoomed out and many points may crowd the screen. Attempting to draw too many points at once may cause the user interface to become unresponsive, especially on low-powered and mobile devices.

A common problem encountered within our research centre with previous tools was that uploading WSI files to the system takes a long time due to their large file size. If there is a dropped packet or interruption to the network connection, then the whole file upload must be restarted. To avoid this frustration, a chunked upload feature was implemented. This allows a failed chunk upload to be retried or resumed after a network connection interruption.
Figure 5.3.2: Salsa user interface for point annotation layers uses a display mode (via an additional plugin for leaflet) that groups points together when zoomed out to avoid crowding and reduce CPU rendering overhead, thus keeping the interface responsive.
Feedback from the Salsa prototype was very positive, with the live collaborative editing a particularly popular feature. Salsa’s development also inspired and motivated further development of such tools within the centre and the PathLAKE project.

5.3.2 Aura: Heatmap Visualization

Visualizing the output of an algorithm during research, development, and production is critical for evaluating performance and interpreting results. Two common forms of data generated by algorithms are a list of points indicating detection locations or two-dimensional maps including patch-wise activations or class activation maps (CAMs) indicating the intensity of response from a model, usually saved as a low-resolution image or serialized NumPy array. Using the same slippery map technique described for Salsa, Aura was developed to display these data formats interactively over a WSI (Figure 5.3.3). This proved useful for interactively demonstrating and exploring results when training models using the polled wavelet representation described in section 4.2.

Aura features interactive adjustment of heatmap parameters and applying various colour maps to the data. Several popular colour maps defined in the matplotlib python library are offered to provide users with visual consistency and familiarity when transitioning from a static matplotlib visualization.
Figure 5.3.3: The Aura user interface showing a heatmap of detected cell nucleus locations overlaid on a WSI.
5.3.3 Tango: Rapid Annotation for the Lysto Challenge

When annotating large WSIs, the scale of the image and annotation task can be quite daunting. To prevent the ‘blank page syndrome’ that often occurs when starting, it can be helpful to limit the user’s view to a single patch for annotation. This approach helps to avoid overthinking and unnecessary edits. Furthermore, it is common for datasets to comprise many small extracted regions or ‘patches’ instead of entire WSIs. As such, a simple-to-use tool for annotating small patches was developed during the 2019 Lymphocyte Assessment (Lysto) Hackathon [11], [148] challenge dataset in conjunction with the MICCAI COMPAY Workshop on Computational Pathology.

The Tango user interface was developed to be simple and allow for rapid point annotation. It has keyboard shortcuts for rapid editing on the desktop, in addition to a mobile-optimized layout with large buttons (Figure 5.3.4) was used so that it works well across many devices. This allows users to choose which device form factor works best for them.

The PathLAKE consortium organized a web-based competition called ‘Beat the Pathologists,’ [149] which involved using the Tango user interface to collect data from students. This event led to the publication of a scientific paper titled ”Lessons from a Breast Cell Annotation Competition Series for School Pupils” in Scientific Reports [150].
Figure 5.3.4: Screenshot showing the Tango user interface. Point annotations are displayed as green circles overlaid on an image patch containing blue haematoxylin-stained and brown DAB-stained cell membranes (CD3 and CD8).
5.4 Chapter Summary

This chapter discusses the challenges of storing annotation data and presents a storage system for efficiently handling annotations. TIA-Toolbox includes two implementations, one using an in-memory dictionary and another using an SQLite database with optimizations for fast queries. With query methods tailored for common use cases and a specialized Python subset DSL, complex queries can be executed efficiently without the user having to understand backend intricacies. Furthermore, this abstraction enables the integration of new backend stores, such as an alternate spatial database, without necessitating any changes to the query logic for the new system.

This chapter also introduces web-based tools for annotation and visualization, including the Salsa application for collaborative annotation, the Aura application for heatmap visualization, and the Tango application for rapid annotation. These tools address the challenges of gathering annotations and provide efficient ways to handle and visualize annotations. Overall, this chapter outlines the importance of annotation in digital pathology and provides practical solutions to enhance the annotation process.

In conclusion, this chapter highlights the challenges of gathering annotation data in digital pathology and presents an efficient storage system and web-based tools for annotation and visualization. These tools improve the annotation process and facilitate collaboration between pathologists and researchers.
Chapter 6

Conclusions

In this chapter, we will summarize the findings and contributions presented in the previous chapters. We discuss the impact of the work on the field of computational pathology, highlight limitations, and suggest future work that could build upon the advancements made in this thesis.

This thesis presented a comprehensive toolbox for computational pathology (TIAToolbox), a tool for fast whole slide image conversion (WSIC), explored the impact of image compression on deep learning models, investigated the use of representations of histology images, and addressed the challenges of annotating histology images. The impact is presented, including the number of downloads, citations, and social media engagement it has received.

The advancements presented in this thesis have the potential to improve the accuracy and efficiency of computational histopathology which may ultimately lead to better patient outcomes in future.
6.1 TIAToolbox

TIAToolbox is an open-source Python library that provides a comprehensive set of tools for computational pathology image analysis. It addresses the need for a unified software library in this field by offering a streamlined and reproducible platform for researchers to construct and reproduce analytical pipelines using state-of-the-art methods. The toolbox includes functionalities for WSI reading, patch extraction, stain normalization, model inference, post-processing, and visualization. It supports a wide range of WSI formats and provides pre-trained models for tasks such as patch classification, semantic segmentation, and nucleus instance segmentation and classification. The toolbox is designed to be user-friendly, with a simple API and extensive documentation, making it accessible to researchers with little expertise in deep learning. It also features example notebooks and a command-line interface.

TIAToolbox contributes to the field of computational pathology by providing a valuable resource for researchers to develop and apply advanced image analysis techniques to large-scale histopathology datasets. Furthermore, TIAToolbox has been used to reproduce two state-of-the-art AI pipelines in computational pathology: IDaRS [37] and SlideGraph+ [62]. These demonstrations highlight the utility of TIAToolbox in simplifying the implementation of complex computational pathology tasks, reducing the amount of code required, and providing reproducible results. The toolbox
enables researchers to focus on **model development** and analysis, rather than the technical details of working with large-scale WSIs. TIAToolbox has the potential to **accelerate the development of new methods and applications** in computational pathology, and its modular design allows for easy integration of new models and functionalities in the future.

The TIAToolbox paper has been published in Nature Medicine Communications (NatMC) on September 24, 2022, following a preprint publication on bioRxiv. It has received seventeen citations (Google Scholar [151]). The toolbox also has been **downloaded over 140,000 times**, with 130,000 via the Python Package Index (PyPI), and over 10,000 on Conda Forge. The TIAToolbox GitHub repository has received **over 200 stars and 50 forks**, and contributions from four external accounts. The toolbox has also been **used to create demonstration web applications accessible at https://tiademos.dcs.warwick.ac.uk/** for visualizing the **output of published models** as companions to manuscripts.

The paper has been tweeted 32 times by 22 accounts. PlumX analytics from Elsevier reports that the paper is in the 86th percentile for first-year tweets, and the paper has been captured on Mendeley 34 times, placing it in the 92nd percentile for first-year captures [152]. This output is placed in the top 25% of all research outputs scored by Altmetric with a high attention score of 18, corresponding to the 90th percentile of all articles of a similar age [153]. These metrics suggest that the TIA Toolbox paper has significantly impacted the academic community, with high downloads, citations.
and social media engagement.

As of writing, the TIA Toolbox paper has received over 4,000 paper accesses. The authors have received positive feedback in person from industry professionals during events like the 18th European Congress on Digital Pathology (ECDP) 2022 in Manchester, UK, and the 19th ECDP 2023 in Budapest, Hungary.

The poster on TIAToolbox was awarded the runner-up prize for best poster at the Alan Turing National Showcase for Data Science and AI Poster Competition ‘What’s Your AI Superpower?’ event on September 12 2022, at the Univerity of Warwick. TIAToolbox has also been featured in the PyTorch Ecosystem [154].

We plan to continue expanding the toolbox with new features and tools. Specifically, we plan to:

- **Integrate new state-of-the-art models:** We plan to add more models to TIAToolbox, keeping up with the latest advances in computational pathology.

- **Enhancing documentation and tutorials:** We will continue providing and expanding upon our detailed documentation that helps to familiarise users with the framework quickly.

- **Broaden the platform and APIs:** We intend to include support for additional imaging formats, including multi-channel fluorescence imaging, to widen TIAToolbox’s scope further. We have created a proof of concept method which dynamically links with optional dependencies for non-CPython native libraries at
runtime such as BioFormats\textsuperscript{1}. This could be refined and made available to open up access to using the toolbox many more image formats via a Java bridge.

- **Integration with other tools:** The toolbox may be optionally integrated with other tools in the area such as QuPath and Cytomine.

- **Refactor the inference engine architecture:** We plan to overhaul the existing inference engine to create a greater decoupling between PyTorch models and the orchestration of data loading, pre-processing, and post-processing. This aims to make future development and integration with other tools easier.

6.2 Whole Slide Image Conversion

WSIC demonstrated leading conversion speed between many WSI formats and offers improved handling of large JP2 images compared to other tools, as well as preservation of vital resolution metadata (MPP). In conversion benchmarks, it performed fastest, or near to the fastest performing tool, in every scenario. WSIC also demonstrated suitable speed for rapidly converting large-scale datasets. We anticipate the digital and computational pathology community to use this tool, contribute to its further development, and expand its use cases.

In the future, WSIC could be expanded to support a wider range

\textsuperscript{1}Available at https://gist.github.com/John-P/7da4224d76eedfd6a5f6239589acaf2f
of image formats. This could include support beyond RGB bright-field images, such as for floating point and multichannel fluorescence imaging which are currently not supported. Additionally, more robust validation and type guarantees could be introduced using a static type-checking system such as mypy or data modelling and validation libraries such as Pydantic. These could help to eliminate and prevent bugs in the conversion code giving great confidence in correct conversion.

Experiments showed that JPEG 2000 and JPEG XL performed significantly better than JPEG 1 and JPEG-LS at the same compression ratio for nucleus segmentation. When considering compression options for WSIs, it appears that either JPEG 2000 or JPEG XL are better choices than JPEG 1, despite the slightly longer encode and decode times associated with increased computation. Additionally, when testing nucleus classification, re-compressing with JPEG 1 compression demonstrated significant deviation from inference on the original even when the highest quality quantisation settings were used. Other codecs which can perform near or perceptually lossless compression with a PSNR of over 40 dB performed significantly better on the task of nucleus classification.

One limitation is that all of the original WSIs obtained from TCGA were compressed with either JPEG 1 or JPEG 2000 irreversible compression codecs. Re-compressing using the same or a different codec may not show the same behaviour as compressing the original uncompressed image data. For example, a codec may
be particularly effective at perceptual compression of high-frequency noise, which may have been lost when the original image was compressed from the scanner. Running the same experiment with raw uncompressed images obtained from the scanner could eliminate this confounding factor. However, this data is difficult to obtain.

Another limitation of the research includes the focus on specific tasks and hardware configurations, and further research could explore larger-scale conversion tasks and the impact of compression on different models and pathology tasks.

Lastly, further work could add support for lossless transformations of some WSIs. For example, the jpegtran utility included with the libjpeg library can flip and rotate JPEG images without error. This could be applied to each tile in an image, along with re-ordering of tiles to flip or rotate an entire whole slide image reversibly. It is also possible to adjust the data organisation and entropy coding method used within a JPEG codestream. This includes changing DCT coefficients to be in a progressive ordering, optimizing Huffman tables, or swapping from Huffman coding to arithmetic coding. Each of which can reduce the file size without altering image quality. Some of this work has already been developed on a branch of the GitHub repository\(^2\) but has not yet been included in the main branch and released version.

Integration with distributed chunked array processing libraries Dask [155] and Xarray [156], [157] is preliminarily supported for TIFF files, Zarr arrays, and NGFF. This integration and support

\(^2\)Accessible at https://github.com/John-P/wsic/tree/feature-arithmetic-jpeg
could be improved to provide easy parallel processing of WSIs. This could also be integrated with ML libraries and the TIAToolbox.

At the time of writing, WSIC has **over 10,000 downloads via PyPI**. It has also received engagement from the open-source community on GitHub including thirteen stars, four forks, one external pull request, and several issues. WSIC also proved an invaluable tool in my research, such as running experiments on the impact of compression (3.7). It has also been used by others with the research centre, such as for converting slow-to-decode JP2 images into pyramid TIFFs as a pre-processing step to speed up data loading and make the images easier to work with using preferred tools.

### 6.3 Leveraging Alternative Representations of Histology Images

This chapter explored using two alternative representations of histology images for model training and inference: a **self-supervised learnt representation** and a **compressed pooled multi-level wavelet representation**.

The chapter demonstrated the utility and generality of the self-supervised learning method, Swapped Augmented Vectors (SwAV), when applied to histology images. By leveraging a large dataset of unlabelled histology images, **SwAV was able to learn generic representations that captured important primitives associated with computational pathology**. These representations proved highly informative, as shown by their ability to accurately
classify tissue types in the validation set. The SwAV embeddings exhibited strong clustering patterns corresponding to the ground truth labels, indicating that the learned representations captured meaningful differences between tissue types. Furthermore, these features transferred via a simple projection head to prediction of cellular composition using only weak labels, which were competitive with specialised models using highly detailed pixel-level labels. This highlights the potential of SwAV as a powerful tool for self-supervised feature representation in histology image analysis. Furthermore, as SwAV is a general learning scheme, it can be applied to various other tasks in computational pathology, providing a versatile and efficient solution. Overall, this work contributes to the field by showcasing the effectiveness of unsupervised representation learning in histology images and underscores the value of SwAV as a generative and informative technique for computational pathology.

However, this self-supervised learning and other patch-wise methods limit the contextual information available to the model during training. To overcome this limitation, we proposed using a pooled multi-level wavelet compression to create a more compressed representation of the images. This compressed representation allows for training on larger visual fields, incorporating context from a larger area. The proposed method was evaluated on tasks such as tissue classification and colorectal adenocarcinoma grading and achieved comparable or better accuracy compared to existing patch-based and context-aware methods. The results
also showed that the proposed method reduces the input data size without significantly impacting accuracy or training time. Overall, this work aimed to contribute a valuable approach to addressing the challenge of working with large WSIs in computational pathology, enabling the use of larger visual fields and improving the contextual information available to CNN models.

However, there are limitations of this research to consider. The self-supervised method had lower performance than full pixel segmentation-based methods for cellular composition prediction, and improvements are needed for predicting irregularly shaped cells. The evaluation was limited to breast tissue images, and further validation is required for generalizability. Additionally, the compressed wavelet method slightly increases training time, but the benefits of larger batch sizes and input visual fields may outweigh this drawback.

Future work could extend the proposed pooled DWT method to other clinical problems requiring a larger context, such as prostate tumour grading or classification of an entire core from a tissue microarray using a single network inference. For an even larger context, aggregation across visual fields or high-level context learning could be employed. Additionally, introducing more extensive and domain-specific augmentation for self-supervised representation learning could further improve the performance of the method.
6.4 Annotating Histology Images

In this chapter, we discussed the challenges associated with gathering annotation data in digital pathology. Annotation is a critical component in training machine learning models and improving the accuracy and speed of diagnosis. However, gathering annotation data can be time-consuming, labour-intensive, and require consensus from multiple pathologists. Ensuring the quality of annotation data is also challenging due to subjectivity and biases. Additionally, handling millions of annotations generated by automated models can be storage and memory-intensive.

To address these challenges, we proposed a storage system for annotation data in TIAToolbox. This system includes two implementations: DictionaryStore for small annotation sets and SQLiteStore for large-scale annotation datasets. The SQLite-based storage class is implemented with a spatial index and optimized utility functions, enabling efficient querying of millions of annotations. We also explored different formats and compression methods for storing annotation data and selected Zlib compressed well-known binary (WKB) format for efficient storage. The performance of the SQLiteStore was demonstrated to be significantly faster than alternative implementations.

In addition to the storage system, we have developed web-based tools for annotation and visualization. Salsa is a web-based collaborative annotation tool that allows pathologists and researchers to gather and collaborate on annotation data remotely. Aura is a vi-
sualization tool that allows researchers to visualize and explore the output of algorithms on WSIs interactively. These tools address the challenges of gathering annotation data and provide efficient ways to handle and visualize annotations.

In conclusion, this chapter highlighted the importance of annotation in digital pathology and presents practical solutions to address the challenges associated with gathering annotation data. The storage system and web-based tools enhance the annotation process and facilitate collaboration among pathologists and researchers. These tools contribute to the advancement of digital pathology and the development of more accurate and efficient diagnosis methods.

6.5 Concluding Remarks

This thesis has made significant contributions to the field of computational pathology. The development of TIAToolbox provides researchers with a comprehensive set of tools for computational pathology image analysis, making it easier and more efficient to develop and apply advanced image analysis techniques to large-scale histopathology datasets. The toolbox has been well-received by the academic community, with high download rates, citations, and engagement on social media platforms. The research on whole slide image conversion has resulted in the development of WSIC, a tool that offers leading conversion speeds. This chapter also investigated the impact of recompressing images on deep-learning model inference for nucleus classification. The research on leveraging alternative
representations of histology images has demonstrated the effectiveness of self-supervised learning and compressed pooled multi-level wavelet representations. Finally, the research on annotating histology images has addressed the challenges associated with gathering, visualising, storing and querying of large annotation datasets.

Overall, the contributions of this thesis have advanced the field of computational pathology and have the potential to have a significant impact. Continued development and application of the tools and techniques presented in this thesis will further enhance the ability to analyze histopathology images and improve the accuracy and efficiency of pathology diagnosis. Future work could include expanding the functionality and capabilities of TIAToolbox, further improving the conversion and compression methods for whole slide images, exploring additional alternative representations of histology images, and developing more advanced annotation tools and techniques. By continuing to innovate and collaborate with the pathology community, we can move towards the goal of leveraging computational methods to improve the understanding and treatment of disease.
Bibliography


breast-cancer-pathology-images-by-combining-handcrafted/10.1117/1.JMI.1.3.034003.full (visited on 10/19/2023).


tern Analysis and Machine Intelligence, issn: 1939-3539. DOI: 10.1109/TPAMI.2019.2936841.


# Implementation without TIAToolbox (16 lines)

```python
import numpy as np
import openslide

PATCH_SIZE = (224, 224)

slide = openslide.OpenSlide("CMU-1-Small-Region.svs")
mpp = np.array([float(slide.properties["openslide.mpp-x"]), float(slide.properties["openslide.mpp-y"])]),
target_mpp = np.ones(2)
scale_factor = target_mpp / mpp
read_patch_size = np.multiply(PATCH_SIZE, scale_factor).astype(int)
mosaic_shape = np.divide(slide.dimensions, read_patch_size).astype(int)

for ij in np.ndindex(tuple(mosaic_shape)):
    xy = (np.multiply(ij, PATCH_SIZE) * scale_factor).astype(int)
    patch = slide.read_region(tuple(xy), 0, tuple(read_patch_size)).convert("RGB")
    patch = patch.resize(PATCH_SIZE)
    if np.percentile(np.mean(patch, axis=-1), 5) > 200:
        continue
    patch.save(f"patches/{' - '.join(str(c) for c in xy)}.jpeg")
```

# TIAToolbox based implementation (5 lines)

```python
from tiatoolbox.tools.patchextraction import SlidingWindowPatchExtractor
from matplotlib import pyplot as plt

extractor = SlidingWindowPatchExtractor(
    "CMU-1-Small-Region.svs",
    (224, 224),
    resolution=1,
    units="mm",
)
for n, patch in enumerate(extractor):
```
Listing A.1: Source code demonstrating how the toolbox can simplify implementation of a common task such as patch extraction. The implementation without using TIAToolbox has 16 lines of code, and the implementation with TIAToolbox has only 5 lines of code. Additionally, the implementation without the toolbox does not work with formats which are not supported by OpenSlide, whereas the version using the TIAToolbox extractor class supports many more WSI file types.

from tiatoolbox.wsicore.wsireader import WSIReader
tissue = WSIReader.open("path/wsi1.svs")
mask = tissue.tissue_mask()
tissue_region, mask_region = (wsi.read_rect(location=(0, 0), size=(512, 512), resolution=0.5, units="mpp",)
    for wsi in (tissue, mask)
)

Listing A.2: A short Python expression demonstrating a synchronous reading of a tissue WSI and a lower resolution mask derived from the tissue WSI. Both read operations use the same function call arguments coordinates, including identical coordinates, as shown on lines 008 through 011. This is despite the underlying image data being stored at different resolutions. This demonstrates the power of the VirtualWSIReader object to enable easy reading from both a WSI and a derived image, such as a tissue mask, which may be internally represented at a different resolution.

from tiatoolbox.models.engine.patch_predictor import PatchPredictor
data = [img1, img2]  # input arrays as a list
predictor = PatchPredictor(pretrained_model="resnet18-kather100k").output = predictor.predict(data, mode="patch")

Listing A.3: An example Python script for patch prediction using TIAToolbox using pre-extracted patches as input.

from tiatoolbox.models.engine.patch_predictor import PatchPredictor
# Input WSI file paths
data = ["path/wsi1.svs", "path/wsi2.svs"]
predictor = PatchPredictor(pretrained_model="resnet18-kather100k")
output = predictor.predict(data,
Listing A.4: An example Python script for patch prediction using TIAToolbox using whole-slide images as input. The same API is used for image tiles by changing the mode to “tile” in the predict method.

```python
from tiatoolbox.models.engine.semantic_segmentor import SemanticSegmentor

# Input WSI file paths
data = ["path/wsi1.svs", "path/wsi2.svs"]

segmentor = SemanticSegmentor(
    pretrained_model="fcn_resnet50_unet-bcss",
)

# WSI prediction
output = segmentor.predict(
    imgs=[wsi_file_name],
    mode="wsi",
)
```

Listing A.5: Supplying a WSI as input to a semantic segmentation model. Here, we use a U-Net with a ResNet50 backbone that is trained on the breast cancer semantic segmentation (BCSS) dataset.

```python
from tiatoolbox.models.engine.nucleus_instance_segmentor import NucleusInstanceSegmentor

# Input WSI file paths
data = ["path/wsi1.svs", "path/wsi2.svs"]

# Instantiate the nucleus instance segmentor
inst_segmentor = NucleusInstanceSegmentor(
    pretrained_model="hovernet_fast-pannuke",
)

# WSI prediction
wsi_output = inst_segmentor.predict(
    [wsi_file_name],
    mode="wsi",
)
```

Listing A.6: Supplying a WSI to a nucleus instance segmentation model. Here, we use a HoVer-Net trained on the PanNuke dataset.

```python
from tiatoolbox.models.engine.patch_predictor import PatchPredictor

predictor = PatchPredictor(
    pretrained_model="resnet18-kather100k",
    pretrained_weights="path/weights.pth",
)
```

Listing A.7: Code showing Python script of how to use your own pretrained weights, rather than those provided by TIAToolbox.

```python
import numpy as np
import matplotlib.pyplot as plt

import SlideGraphConstructor
```
# Load XY patch positions in the WSI
positions = np.load(f"path/wsi1.position.npy")
# Load features for each patch
features = np.load(f"path/wsi1.features.npy")
constructor = SlideGraphConstructor()
graph = constructor.build(
    positions[:, :2],
    features,
    feature_range_thresh=None,
)
constructor.visualise(graph)
plt.show()

Listing A.8: Building and visualizing a slide graph from a set of patch locations and associated features.

from tiatoolbox.wisecore.wsireader import WSIReader
from tiatoolbox.visualization.tileserver import TileServer

wsi = WSIReader.open("path/wsi1.svs")
app = TileServer(
title = "Testing TileServer",
layers = {
    "wsi1": wsi,
},
)
app.run()

Listing A.9: Creating a tile server web server gateway interface (WSGI) application and running a simple development HTTP server to display two WSI images in a web browser.

from shapely.Geometry import Polygon
from tiatoolbox.annotation.storage import SQLiteStore, Annotation

# Store a simple triangle annotation
store = SQLiteStore("annotations.db")
annotation = Annotation(
    geometry=Polygon([(0, 0), (1, 0), (0, 1)]),
    properties={},
)
uuid = store.append(annotation)

# Access the annotation and add some properties
print(store[uuid])
annotation.properties = {"class": 1, "foo": "bar"}
store[uuid] = Annotation(translation)

# Change just part of the properties
store.patch(uuid, {"class": 2})

# Query in a bounding box (left, right, top, bottom)
uuids = store.iquery([0, 0, 1, 1])

# Query with a predicate statement
results = store.query([0, 0, 1, 1], where="props['class']==2")

Listing A.10: An example of creating an annotation and storing it in an SQLite database. The annotation is given a universally unique identifier (UUID) as no
key was specified. This is used to look up the annotation and modify it. Also shown here, is how to query for annotations using a bounding box and adding a predicate statement to filter the results.

```python
import numpy as np

from tiatoolbox.models.engine.patch_predictor import PatchPredictor
from tiatoolbox.utils.misc import imwrite

WSI_PATH = "path wsii.svs"

# Tumour detection

pretrained_model = "resnet18-idars-tumour"

tumour_predictor = PatchPredictor()
tumour_output = tumour_predictor.predict(imgs=[WSI_PATH],
 mode='wsi',
)

tumour_mask = tumour_predictor.merge_predictions(
WSI_PATH,
tumour_output[0],
resolution=5,
units="power",
)
tumour_mask = tumour_mask == 2 # Binarize the output
imwrite("tumour_mask.png", tumour_mask.astype("uint8"))

# WSI Prediction

pretrained_model = "resnet34-idars-msi",

msi_predictor = PatchPredictor()
msi_output = msi_predictor.predict(imgs=[WSI_PATH],
mode='wsi',
return_probabilities=True,
)

# Slide-Level Score:

# Only consider MSI class
msi_probabilities = np.array(msi_output[0]['probabilities'][...1]
# Get the average over all tumour tiles
average_msi_probability = np.mean(msi_probabilities)
```

Listing A.11: IDaRS inference using TIAToolbox. Here, we demonstrate that we can simplify the overall inference pipeline without the need for many lines of code. The following steps are performed: 1) tumor detection, 2) saving tumor mask, 3) mutation prediction in tumor regions and 4) patch aggregation.
## B  Tables

<table>
<thead>
<tr>
<th>Model Family</th>
<th>Architecture Variants</th>
<th>Training Dataset(s)</th>
<th>Metric(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet [42]</td>
<td>AlexNet</td>
<td>Kather 100k, PCam</td>
<td>Appendix B.2</td>
</tr>
<tr>
<td>ResNeXt [158]</td>
<td>ResNeXt-50, ResNeXt-101</td>
<td>Kather 100k, PCam</td>
<td>Appendix B.2</td>
</tr>
<tr>
<td>DenseNet [76]</td>
<td>DenseNet121, DenseNet161, DenseNet169, DenseNet201</td>
<td>Kather 100k, PCam</td>
<td>Appendix B.2</td>
</tr>
<tr>
<td>MobileNet [160], [161]</td>
<td>MobileNet-v2, MobileNet-v3 small, MobileNet-v3 large</td>
<td>Kather 100k, PCam</td>
<td>Appendix B.2</td>
</tr>
<tr>
<td>GoogLeNet [129]</td>
<td>GoogLeNet</td>
<td>Kather 100k, PCam</td>
<td>Appendix B.2</td>
</tr>
</tbody>
</table>

Table B.1: Detailed metrics of Models for patch prediction.
### Table B.2: Patch classification performance of models provided by TIAToolbox on both the Kather100K (NCT-CRC-HE-100K) and Patch CAMELYON (PCam) datasets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Kather100K $F_1$</th>
<th>PCam $F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet [42]</td>
<td>0.965</td>
<td>0.840</td>
</tr>
<tr>
<td>ResNet18 [75]</td>
<td>0.990</td>
<td>0.888</td>
</tr>
<tr>
<td>ResNet34 [75]</td>
<td>0.991</td>
<td>0.889</td>
</tr>
<tr>
<td>ResNet50 [75]</td>
<td>0.989</td>
<td>0.892</td>
</tr>
<tr>
<td>ResNet101 [75]</td>
<td>0.989</td>
<td>0.888</td>
</tr>
<tr>
<td>ResNeXt50 32×4d [158]</td>
<td>0.992</td>
<td>0.900</td>
</tr>
<tr>
<td>ResNeXt101 32×8d [158]</td>
<td>0.991</td>
<td>0.892</td>
</tr>
<tr>
<td>Wide ResNet50 [159]</td>
<td>0.989</td>
<td>0.901</td>
</tr>
<tr>
<td>Wide ResNet101 [159]</td>
<td>0.990</td>
<td>0.898</td>
</tr>
<tr>
<td>DenseNet121 [76]</td>
<td>0.993</td>
<td>0.897</td>
</tr>
<tr>
<td>DenseNet161 [76]</td>
<td>0.992</td>
<td>0.893</td>
</tr>
<tr>
<td>DenseNet169 [76]</td>
<td>0.992</td>
<td>0.895</td>
</tr>
<tr>
<td>DenseNet201 [76]</td>
<td>0.991</td>
<td>0.891</td>
</tr>
<tr>
<td>GoogLeNet [129]</td>
<td>0.990</td>
<td>0.899</td>
</tr>
<tr>
<td>MobileNet v2 [160]</td>
<td>0.991</td>
<td>0.895</td>
</tr>
<tr>
<td>MobileNet v3 large [161]</td>
<td>0.992</td>
<td>0.890</td>
</tr>
<tr>
<td>MobileNet v3 small [161]</td>
<td>0.992</td>
<td>0.867</td>
</tr>
</tbody>
</table>

Table B.3: Semantic segmentation performance (Sørensen–Dice score) on the Breast Cancer Semantic Segmentation (BCSS) dataset. Here, we use a U-Net model with a ResNet50 encoder.

<table>
<thead>
<tr>
<th>Tumor</th>
<th>Stroma</th>
<th>Inflammatory</th>
<th>Necrosis</th>
<th>Other</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amgad et al. [80]</td>
<td>0.851</td>
<td>0.800</td>
<td>0.712</td>
<td>0.723</td>
<td>0.666</td>
</tr>
<tr>
<td>TIAToolbox [162]</td>
<td>0.885</td>
<td>0.825</td>
<td>0.761</td>
<td>0.765</td>
<td>0.581</td>
</tr>
</tbody>
</table>

228
Table B.4: Performance of IDaRS provided as part of TIAToolbox, compared to the original implementation.

<table>
<thead>
<tr>
<th>Model</th>
<th>MSI</th>
<th>TP53</th>
<th>BRAF</th>
<th>CIMP</th>
<th>CIN</th>
<th>HM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bilal et al. [37]</td>
<td>0.828</td>
<td>0.755</td>
<td>0.813</td>
<td>0.853</td>
<td>0.860</td>
<td>0.846</td>
</tr>
<tr>
<td>TIAToolbox [162]</td>
<td>0.870</td>
<td>0.747</td>
<td>0.750</td>
<td>0.748</td>
<td>0.810</td>
<td>0.790</td>
</tr>
</tbody>
</table>

Table B.5: Performance of SlideGraph+ using five-fold cross-validation provided as part of TIAToolbox, compared to the original implementation. Here we report the mean and the standard deviation across the folds.

<table>
<thead>
<tr>
<th>Model</th>
<th>HER2</th>
<th>ER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lu et al. (original) [62]</td>
<td>0.710±0.020</td>
<td>-</td>
</tr>
<tr>
<td>TIAToolbox [162]</td>
<td>0.738±0.043</td>
<td>0.872±0.023</td>
</tr>
</tbody>
</table>

Table B.6: Detailed metrics of Models for semantic segmentation.

<table>
<thead>
<tr>
<th>Model Family</th>
<th>Architecture</th>
<th>Variants</th>
<th>Training Dataset(s)</th>
<th>Metric(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unet [40]</td>
<td>ResNet-50</td>
<td>backbone</td>
<td>TCGA-BCSS</td>
<td>Appendix B.3</td>
</tr>
<tr>
<td>HoVer-Net</td>
<td>HoVer-Net+</td>
<td></td>
<td>Private oral dysplasia cohort (not available)</td>
<td>Shephard et al.</td>
</tr>
</tbody>
</table>

Table B.7: Detailed metrics of Models for nucleus segmentation or classification.

<table>
<thead>
<tr>
<th>Model Family</th>
<th>Architecture</th>
<th>Training Dataset(s)</th>
<th>Metric(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HoVer-Net</td>
<td>Original [61], Fast[163], HoVer-Net+ [79]</td>
<td>Kumar (MoNuSeg Subset), PanNuke, CoNSeP, MoNuSAC, Private oral dysplasia cohort (not available)</td>
<td>Graham et al. [61], Gamper et al. [84], [85], Shephard et al. [79]</td>
</tr>
<tr>
<td>Model Family</td>
<td>Architecture Variants</td>
<td>Training Dataset(s)</td>
<td>Metric(s)</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------</td>
<td>---------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>IDaRS [37]</td>
<td>ResNet-18 (tumor),</td>
<td>TCGA-COAD</td>
<td>Appendix B.4</td>
</tr>
<tr>
<td></td>
<td>ResNet-34 (mutation)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SlideGraph [62]</td>
<td>SlideGraph+</td>
<td>TCGA-BRCA</td>
<td>Appendix B.5, Lu et al.</td>
</tr>
</tbody>
</table>

Table B.8: Whole slide classification with SlideGraph and IDaRS.