

AI in Microscopy: A Biolmaging Guide

Table of contents

Welcome	6
1 Introduction	7
1.1 Include section headers as appropriate	7
1.1.1 Sub-subsection headers are also available	7
1.2 Bibliography and Citations	8
1.3 Adding to the Glossary	8
1.4 Code and Equations	8
1.4.1 Embedding Figures	10
1.5 Quarto has additional features.	12
I Getting Started with AI	13
2 AI Primer	14
2.1 Include section headers as appropriate	14
2.1.1 Sub-subsection headers are also available	14
2.2 Bibliography and Citations	15
2.3 Adding to the Glossary	15
2.4 Code and Equations	15
2.4.1 Embedding Figures	17
2.5 Quarto has additional features.	19
3 Foundations of Large Language Models	20
3.1 Multi-modal AI: Vision-Language Models and Generative AI	20
3.2 AI Agents for Microscopy Workflows	20
3.3 Challenges and Limitations	21
3.4 Future Directions	21
3.5 Practical Guide: Getting Started with LLMs for Microscopy	21
4 Architectures and Loss Models	22
4.1 Include section headers as appropriate	22
4.1.1 Sub-subsection headers are also available	22
4.2 Bibliography and Citations	22
4.3 Adding to the Glossary	23

4.4	Code and Equations	23
4.4.1	Embedding Figures	25
4.5	Quarto has additional features.	26
II	Image Acquisition	27
5	Collecting Training Data	28
5.1	Include section headers as appropriate	28
5.1.1	Sub-subsection headers are also available	28
5.2	Bibliography and Citations	28
5.3	Adding to the Glossary	29
5.4	Code and Equations	29
5.4.1	Embedding Figures	31
5.5	Quarto has additional features.	32
6	Extending Your Hardware With AI	33
6.1	Include section headers as appropriate	33
6.1.1	Sub-subsection headers are also available	33
6.2	Bibliography and Citations	34
6.3	Adding to the Glossary	34
6.4	Code and Equations	34
6.4.1	Embedding Figures	36
6.5	Quarto has additional features.	38
7	Adding AI to Your Hardware	39
III	Image Analysis	41
8	Chapter 8: How do you select and find a tool?	42
8.1	Introduction	42
8.2	Assessing your requirements for a tool (in a perfect world)	42
8.2.1	Questions about your overall workflow:	44
8.2.2	Questions about your resources: time, expertise, and compute:	44
8.2.3	Questions about your input data:	45
8.3	Assessing your requirements for a tool (in the real world)	47
8.4	How do I decide when my workflow is “good enough”?	48
8.5	Choosing the kind of tool to use	50
8.6	Finding tools and models	55
8.6.1	Where to look for tools and models	55
8.7	Case Studies	56
8.7.1	Case Study 1:	56
8.7.2	Case Study 2:	60

8.8 Conclusion	65
9 How to Train and Use Deep Learning Models in Microscopy	66
9.1 Introduction	66
9.1.1 The challenge	66
9.2 Preparing for your project	67
9.2.1 Defining your task and success criteria	67
9.2.2 Evaluating alternatives: Is DL the right choice?	68
9.3 Implementing a DL segmentation workflow	70
9.3.1 Overview of a typical DL segmentation workflow	70
9.3.2 Selecting a suitable DL approach	71
9.3.3 Deciding whether to train a new model	71
9.3.4 Preparing your dataset for training	73
9.3.5 Training a segmentation model from scratch	76
9.3.6 Fine-Tuning Pre-existing Models	79
9.3.7 Evaluating the performance of your model	81
9.3.8 Deploying your model on new data	84
9.3.9 Troubleshooting Common Problems	84
9.4 Further considerations for DL segmentation	88
9.4.1 Choosing the Right Tools for DL	88
9.4.2 Managing Computational Resources	89
9.4.3 Ensuring Reproducibility in DL	90
9.5 Summary & Outlook	91
10 Output Quality	93
10.1 Include section headers as appropriate	93
10.1.1 Sub-subsection headers are also available	93
10.2 Bibliography and Citations	93
10.3 Adding to the Glossary	94
10.4 Code and Equations	94
10.4.1 Embedding Figures	96
10.5 Quarto has additional features.	97
11 Outlook	98
11.1 Include section headers as appropriate	98
11.1.1 Sub-subsection headers are also available	98
11.2 Bibliography and Citations	98
11.3 Adding to the Glossary	99
11.4 Code and Equations	99
11.4.1 Embedding Figures	101
11.5 Quarto has additional features.	102

Glossary	103
Backpropagation	103
Batch	103
Bayesian Optimization	103
Binary Segmentation	103
Computer Vision	104
Data Augmentation	104
Domain Randomization	104
Effect Size	104
Epoch	104
F1 Score	104
False Negatives	105
False Positives	105
Gaussian Process	105
Genetic Algorithms	105
Ground Truth	105
Hallucinations	106
Hyperparameters	106
Image Classification	106
Inferences	106
Instance Segmentation	106
IoU	106
Manual Annotation	107
Metadata	107
Object Detection	107
Panoptic Segmentation	107
Pixel Classifiers	107
Quality Control Metric	107
Semantic Segmentation	108
Star-convex Polygon	108
Test Data	108
Training Data	108
Transfer Learning	108
True Negatives	108
True Positives	109
Validation Data	109
Virtual Machine	109
References	110

Welcome

This is an initial outline for the welcome page:

- Include a very brief introduction of the book (this is not the introduction chapter).
- Explain how to interact with the book. For example, explain how code snippets work for the reader, links to figures, glossary terms, etc.
- State the licensing and use restrictions.
- How to cite the book.

1 Introduction

AI at Every Stage of the Microscopy Workflow

Under your first header, include a brief introduction to your chapter.

Starting prompt for this chapter: Chapter 1 outlines how AI can span experimental design, image acquisition, image processing, and analysis (without discussing what AI is from a technical perspective). This chapter will also outline the roadmap of the book which will largely focus on acquisition and processing.

Topics suggested during the authors' meetings: Discuss that AI is not always solution and talk about when it is actually useful. Discuss that there are many types of microscopy images and each will have their own AI considerations (e.g., imaging modality, 2D vs 3D, static vs time lapse).

1.1 Include section headers as appropriate

Use markdown heading level two for section headers. You can use standard markdown formatting, for example *emphasize the end of this sentence*.

This is a new paragraph with more text. Your paragraphs can cross reference other items, such as Figure 11.1. Use `fig` to reference figures, and `eq` to reference equations, such as Equation 11.1.

1.1.1 Sub-subsection headers are also available

To make your sections cross reference-able throughout the book, include a section reference, as shown in the header for Section 11.4.

1.2 Bibliography and Citations

To cite a research article, add it to references.bib and then refer to the citation key. For example, reference Stringer et al. [62] refers to CellPose and reference Chamier et al. [14] refers to ZeroCostDL4Mic.

1.3 Adding to the Glossary

We are using R code to create a glossary for this book. To add a definition, edit the glossary.yml file. To reference the glossary, enclose the word as in these examples: LLMs suffer from **hallucinations**. It is important to understand the underlying **training data**, **validation data** and **false positives** to interpret your results. Clicking on the word will reveal its definition by taking you to the entry on the [Glossary](#) page. Pressing back in your browser will return you to your previous place in the textbook.

1.4 Code and Equations

This is an example of including a python snippet that generates a figure

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

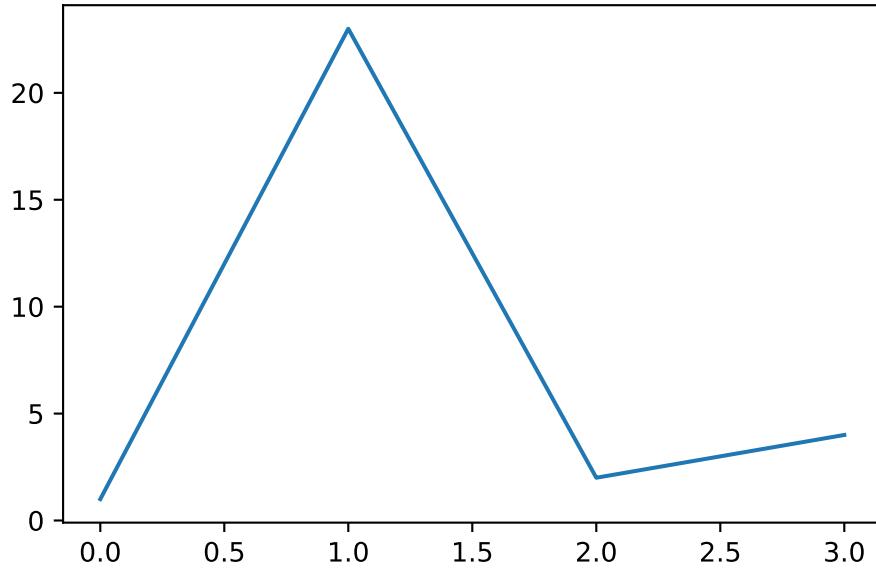


Figure 1.1: Simple Plot

In some cases, you may want to include a code-block that is not executed when the book is compiled. Use the `eval: false` option for this.

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

Figures can also be generated that do not show the code by using the option for `code-fold: true`.

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

r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
fig, ax = plt.subplots(
    subplot_kw = {'projection': 'polar'}
)
ax.plot(theta, r)
ax.set_rticks([0.5, 1, 1.5, 2])
ax.grid(True)
plt.show()
```

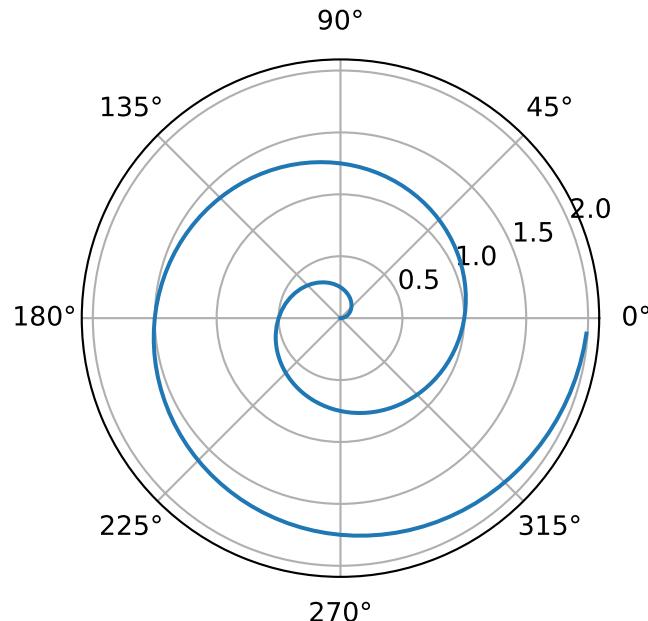


Figure 1.2: A spiral on a polar axis

Here is an example equation.

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (1.1)$$

1.4.1 Embedding Figures

You can also embed figures from other notebooks in the repo as shown in the following embed example.

When embedding notebooks, please store the .ipynb file in the notebook directory. Include the chapter in the name of your file. For example, `chapter4_example_u-net.ipynb`. This is how we will handle chapter- or example-specific environments. We will host notebooks on Google Colab so that any required packages for the code—but not for rendering the book at large—will be installed there. That way, we will not need to handle a global environment across the book.

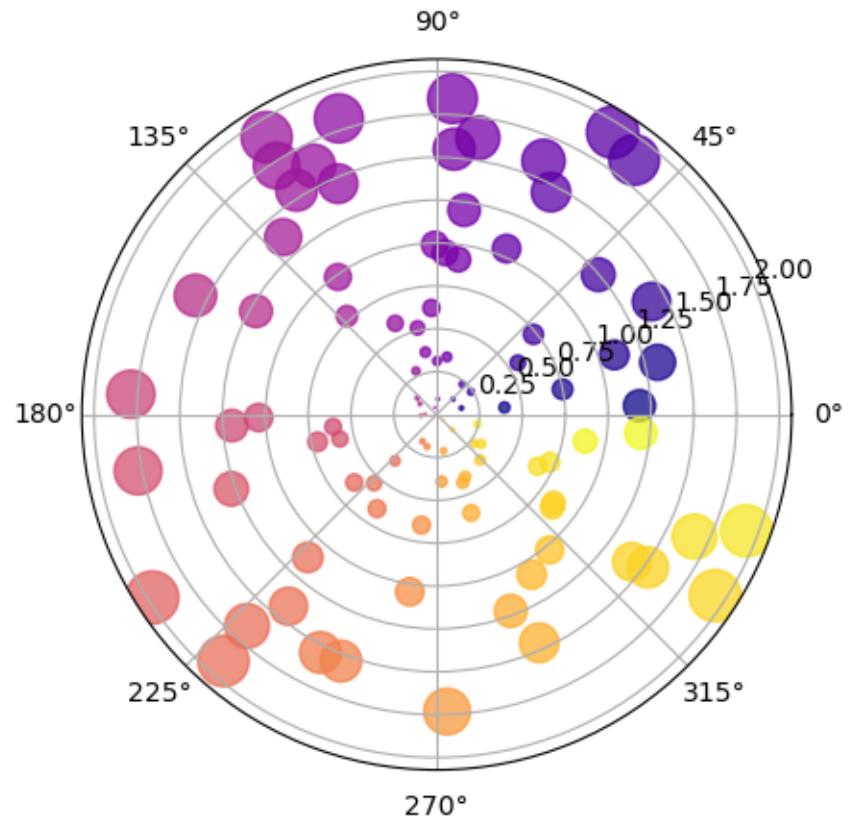


Figure 1.3: Polar plot of circles of random areas at random coords

1.5 Quarto has additional features.

You can learn more about markdown options and additional Quarto features in the [Quarto documentation](#). One example that you might find interesting is the option to include callouts in your text. These callouts can be used to highlight potential pitfalls or provide additional optional exercises that the reader might find helpful. Below are examples of the types of callouts available in Quarto.

Note

Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`. They can default to open (like this example) or collapsed (example below).

Tip

These could be good for extra material or exercises.

Caution

There are caveats when applying these tools. Expand the code below to learn more.

```
r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
```

Warning

Be careful to avoid hallucinations.

Important

This is key information.

Part I

Getting Started with AI

2 AI Primer

An Introduction to Artificial Intelligence

Under your first header, include a brief introduction to your chapter.

Starting prompt for this chapter: Chapter 2 demystifies Artificial Intelligence for microscopy users. It should define terms (e.g., machine/deep learning, supervised/unsupervised learning) without programming details such that an educated scientist without AI experience can understand how these concepts apply to microscopy in life sciences. The use-cases and strengths of different approaches for different applications should be discussed (e.g., contrasting unsupervised clustering vs supervised segmentation). This chapter should broadly introduce image restoration and segmentation, as they will be themes throughout.

Suggestion from authors' meetings: This chapter can draw on the outlines from other chapters to introduce key topics for the following chapters.

2.1 Include section headers as appropriate

Use markdown heading level two for section headers. You can use standard markdown formatting, for example *emphasize the end of this sentence*.

This is a new paragraph with more text. Your paragraphs can cross reference other items, such as Figure 11.1. Use `fig` to reference figures, and `eq` to reference equations, such as Equation 11.1.

2.1.1 Sub-subsection headers are also available

To make your sections cross reference-able throughout the book, include a section reference, as shown in the header for Section 11.4.

2.2 Bibliography and Citations

To cite a research article, add it to references.bib and then refer to the citation key. For example, reference Stringer et al. [62] refers to CellPose and reference Chamier et al. [14] refers to ZeroCostDL4Mic.

2.3 Adding to the Glossary

We are using R code to create a glossary for this book. To add a definition, edit the glossary.yml file. To reference the glossary, enclose the word as in these examples: LLMs suffer from **hallucinations**. It is important to understand the underlying **training data**, **validation data** and **false positives** to interpret your results. Clicking on the word will reveal its definition by taking you to the entry on the [Glossary](#) page. Pressing back in your browser will return you to your previous place in the textbook.

2.4 Code and Equations

This is an example of including a python snippet that generates a figure

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

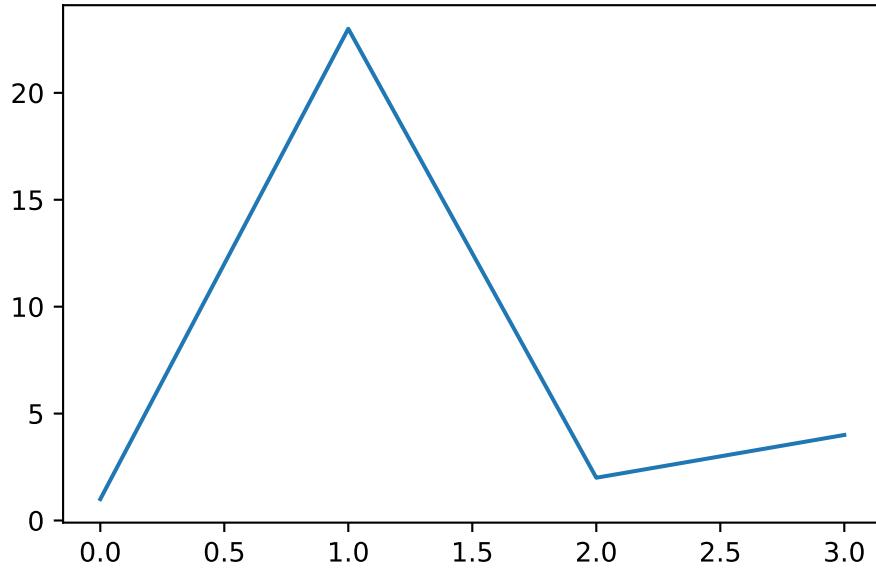


Figure 2.1: Simple Plot

In some cases, you may want to include a code-block that is not executed when the book is compiled. Use the `eval: false` option for this.

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

Figures can also be generated that do not show the code by using the option for `code-fold: true`.

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

r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
fig, ax = plt.subplots(
    subplot_kw = {'projection': 'polar'}
)
ax.plot(theta, r)
ax.set_rticks([0.5, 1, 1.5, 2])
ax.grid(True)
plt.show()
```

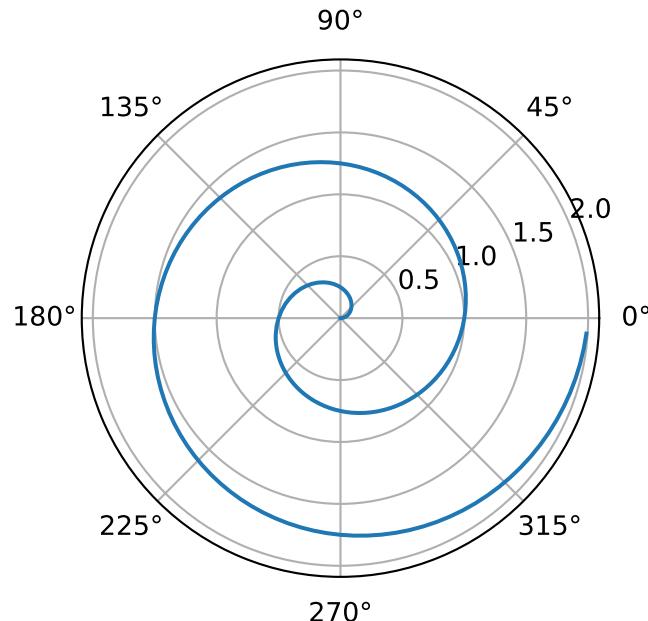


Figure 2.2: A spiral on a polar axis

Here is an example equation.

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (2.1)$$

2.4.1 Embedding Figures

You can also embed figures from other notebooks in the repo as shown in the following embed example.

When embedding notebooks, please store the .ipynb file in the notebook directory. Include the chapter in the name of your file. For example, `chapter4_example_u-net.ipynb`. This is how we will handle chapter- or example-specific environments. We will host notebooks on Google Colab so that any required packages for the code—but not for rendering the book at large—will be installed there. That way, we will not need to handle a global environment across the book.

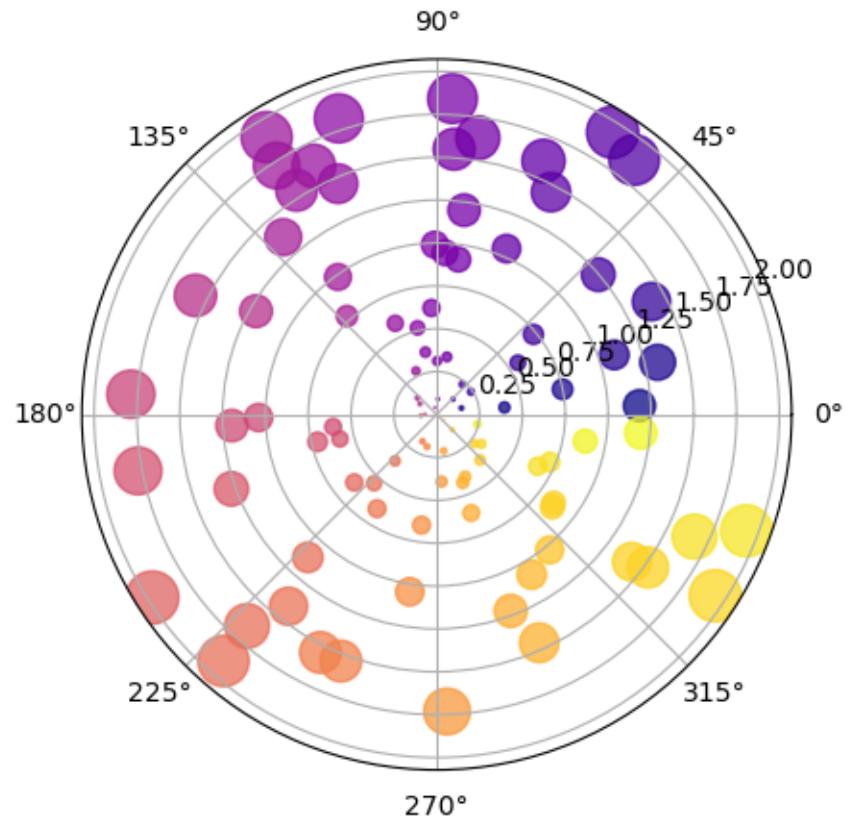


Figure 2.3: Polar plot of circles of random areas at random coords

2.5 Quarto has additional features.

You can learn more about markdown options and additional Quarto features in the [Quarto documentation](#). One example that you might find interesting is the option to include callouts in your text. These callouts can be used to highlight potential pitfalls or provide additional optional exercises that the reader might find helpful. Below are examples of the types of callouts available in Quarto.

Note

Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`. They can default to open (like this example) or collapsed (example below).

Tip

These could be good for extra material or exercises.

Caution

There are caveats when applying these tools. Expand the code below to learn more.

```
r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
```

Warning

Be careful to avoid hallucinations.

Important

This is key information.

3 Foundations of Large Language Models

Large Language Models and AI Agents for Microscopy Imaging

In recent years, large language models (LLMs) have revolutionized how we interact with technology, bringing unprecedented capabilities to scientific research including microscopy. This chapter explores how microscopists can leverage these powerful AI tools to enhance their workflow, from learning concepts to automating analysis tasks. We'll discuss both general-purpose and microscopy-specific tools while highlighting practical applications and potential pitfalls.

This section introduces the fundamental concepts behind modern language models, focusing on transformer architectures that power tools like ChatGPT. We'll explain how these models function, their capabilities for understanding scientific text, and their emerging role in generating code for image analysis tasks. We'll demonstrate how microscopists can effectively use LLMs to learn new concepts, troubleshoot methods, and generate starting points for analysis scripts.

3.1 Multi-modal AI: Vision-Language Models and Generative AI

Moving beyond text-only interfaces, multi-modal models combine language understanding with visual processing capabilities. This section explores how Vision-Language Models (VLMs) like GPT-4o can “see” and interpret microscopy images, assist with image annotation, and even aid in experimental design. We'll also cover generative AI technologies including diffusion models that can create synthetic training data, perform style transfer, or convert microscopy images into vector graphics for publications.

3.2 AI Agents for Microscopy Workflows

AI agents represent the next evolution - autonomous systems that combine language understanding with specialized scientific knowledge and the ability to execute actions. We'll examine microscopy-specific tools like Omega and the BioImage.io chatbot that can perform complex bioimage analysis workflows through natural language instructions. This section will explore

chain-of-thought reasoning, code generation and execution capabilities, and how these agents use visual feedback to iteratively improve results.

3.3 Challenges and Limitations

While powerful, AI assistants come with significant limitations that microscopists must understand. This section addresses critical challenges including:

- and factual errors in generated content
- The “black box” nature of models and concerns about reproducibility
- Alignment problems when tools lack domain-specific knowledge
- The need for human validation and the dangers of overreliance
- Practical strategies for steering models toward scientifically valid outputs

3.4 Future Directions

The intersection of LLMs and microscopy is rapidly evolving. This final section examines emerging capabilities and future possibilities, including:

- Generalist vision-language models capable of performing diverse analysis tasks
- Models that can directly transform input images into processed outputs
- The integration of AI agents with microscope hardware for fully autonomous imaging
- Smart microscopy systems that adapt acquisition parameters based on real-time image understanding
- Ethical considerations and best practices for responsible AI adoption in biological research

3.5 Practical Guide: Getting Started with LLMs for Microscopy

This hands-on section provides step-by-step guidance for microscopists to begin leveraging LLMs effectively, including:

- Crafting effective prompts that produce reliable, scientific outputs
- Using ChatGPT and similar tools to learn imaging concepts and generate analysis code
- Getting started with BioImage.io tools and microscopy-specific AI agents
- Strategies for validating and verifying AI-generated solutions
- Example workflows demonstrating LLM integration into real microscopy analysis tasks

4 Architectures and Loss Models

optionally add a subtitle

Under your first header, include a brief introduction to your chapter.

Starting prompt for this chapter: Chapter 4 introduces architectures and loss models, defining them and providing examples through two practical case studies: image restoration and segmentation. Although this chapter will include code snippets/exercises, the presentation of essential concepts should communicate the philosophy behind the choice of a model for non-programmers.

4.1 Include section headers as appropriate

Use markdown heading level two for section headers. You can use standard markdown formatting, for example *emphasize the end of this sentence*.

This is a new paragraph with more text. Your paragraphs can cross reference other items, such as Figure 11.1. Use `fig` to reference figures, and `eq` to reference equations, such as Equation 11.1.

4.1.1 Sub-subsection headers are also available

To make your sections cross reference-able throughout the book, include a section reference, as shown in the header for Section 11.4.

4.2 Bibliography and Citations

To cite a research article, add it to references.bib and then refer to the citation key. For example, reference Stringer et al. [62] refers to CellPose and reference Chamier et al. [14] refers to ZeroCostDL4Mic.

4.3 Adding to the Glossary

We are using R code to create a glossary for this book. To add a definition, edit the glossary.yml file. To reference the glossary, enclose the word as in these examples: LLMs suffer from [hallucinations](#). It is important to understand the underlying [training data](#), [validation data](#) and [false positives](#) to interpret your results. Clicking on the word will reveal its definition by taking you to the entry on the [Glossary](#) page. Pressing back in your browser will return you to your previous place in the textbook.

4.4 Code and Equations

This is an example of including a python snippet that generates a figure

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

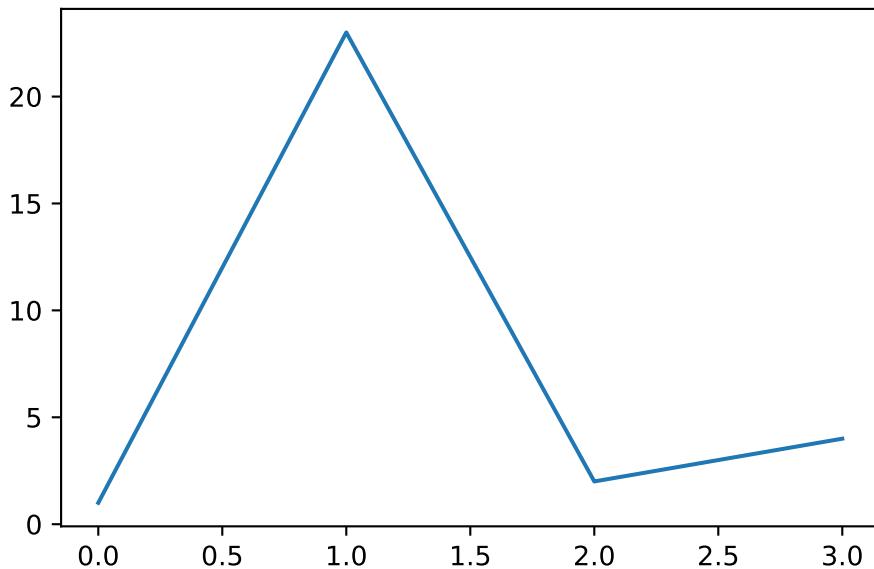


Figure 4.1: Simple Plot

In some cases, you may want to include a code-block that is not executed when the book is compiled. Use the `eval: false` option for this.

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

Figures can also be generated that do not show the code by using the option for `code-fold: true`.

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

r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
fig, ax = plt.subplots(
    subplot_kw = {'projection': 'polar'}
)
ax.plot(theta, r)
ax.set_rticks([0.5, 1, 1.5, 2])
ax.grid(True)
plt.show()
```

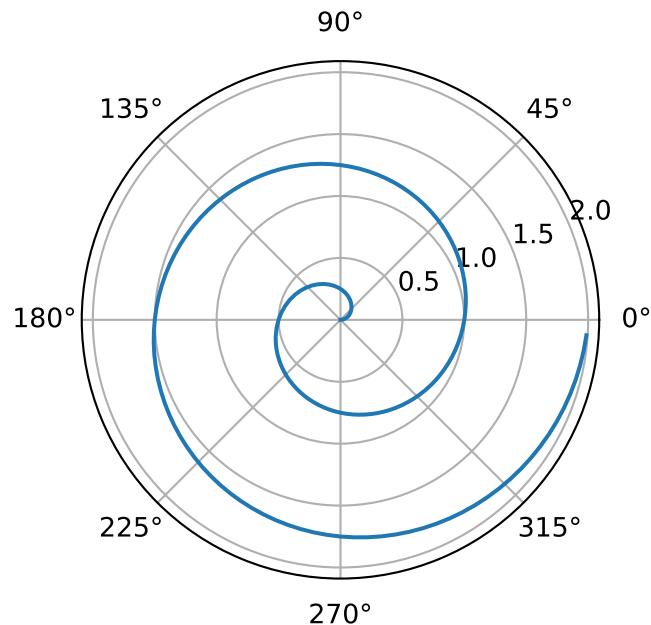


Figure 4.2: A spiral on a polar axis

Here is an example equation.

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (4.1)$$

4.4.1 Embedding Figures

You can also embed figures from other notebooks in the repo as shown in the following embed example.

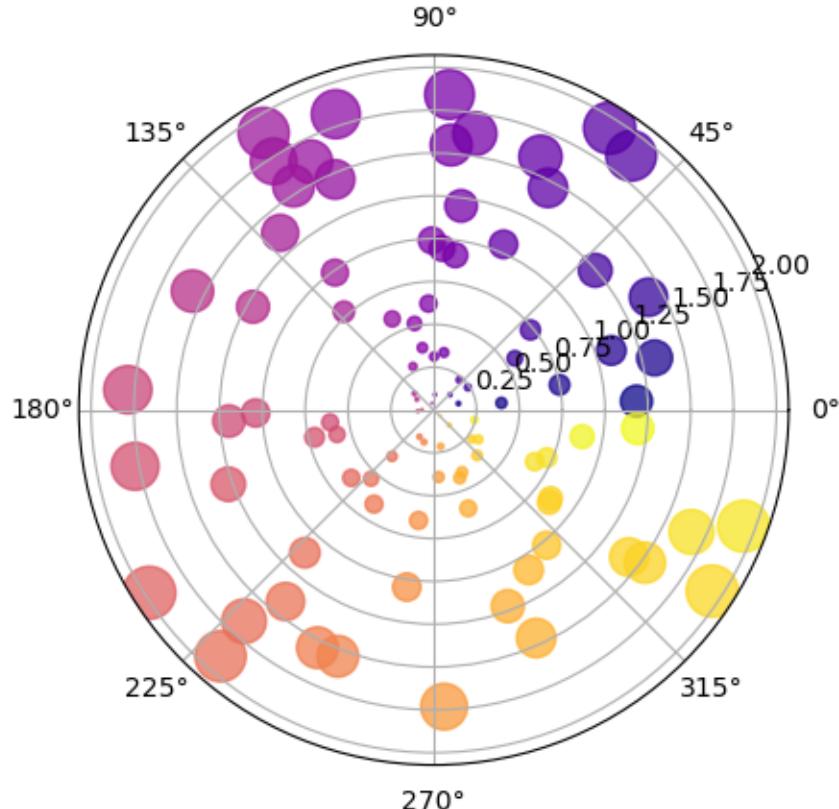


Figure 4.3: Polar plot of circles of random areas at random coords

When embedding notebooks, please store the .ipynb file in the notebook directory. Include the chapter in the name of your file. For example, `chapter4_example_u-net.ipynb`. This is how we will handle chapter- or example-specific environments. We will host notebooks on Google Colab so that any required packages for the code—but not for rendering the book at large—will be installed there. That way, we will not need to handle a global environment across the book.

4.5 Quarto has additional features.

You can learn more about markdown options and additional Quarto features in the [Quarto documentation](#). One example that you might find interesting is the option to include callouts in your text. These callouts can be used to highlight potential pitfalls or provide additional optional exercises that the reader might find helpful. Below are examples of the types of callouts available in Quarto.

Note

Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`. They can default to open (like this example) or collapsed (example below).

Tip

These could be good for extra material or exercises.

Caution

There are caveats when applying these tools. Expand the code below to learn more.

```
r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
```

Warning

Be careful to avoid hallucinations.

Important

This is key information.

Part II

Image Acquisition

5 Collecting Training Data

Image Collection and Considerations

Under your first header, include a brief introduction to your chapter.

Starting prompt for this chapter: Chapter 5 discusses collecting, annotating and validating training data. It should highlight potential pitfalls such as balanced data sets, out-of-distribution problems, etc. It should also address the question: how do you collect training data on your microscope? For example, this chapter should discuss collecting low/high-laser power pairs for the purpose of training an image restoration model.

5.1 Include section headers as appropriate

Use markdown heading level two for section headers. You can use standard markdown formatting, for example *emphasize the end of this sentence*.

This is a new paragraph with more text. Your paragraphs can cross reference other items, such as Figure 11.1. Use `fig` to reference figures, and `eq` to reference equations, such as Equation 11.1.

5.1.1 Sub-subsection headers are also available

To make your sections cross reference-able throughout the book, include a section reference, as shown in the header for Section 11.4.

5.2 Bibliography and Citations

To cite a research article, add it to references.bib and then refer to the citation key. For example, reference Stringer et al. [62] refers to CellPose and reference Chamier et al. [14] refers to ZeroCostDL4Mic.

5.3 Adding to the Glossary

We are using R code to create a glossary for this book. To add a definition, edit the glossary.yml file. To reference the glossary, enclose the word as in these examples: LLMs suffer from [hallucinations](#). It is important to understand the underlying [training data](#), [validation data](#) and [false positives](#) to interpret your results. Clicking on the word will reveal its definition by taking you to the entry on the [Glossary](#) page. Pressing back in your browser will return you to your previous place in the textbook.

5.4 Code and Equations

This is an example of including a python snippet that generates a figure

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

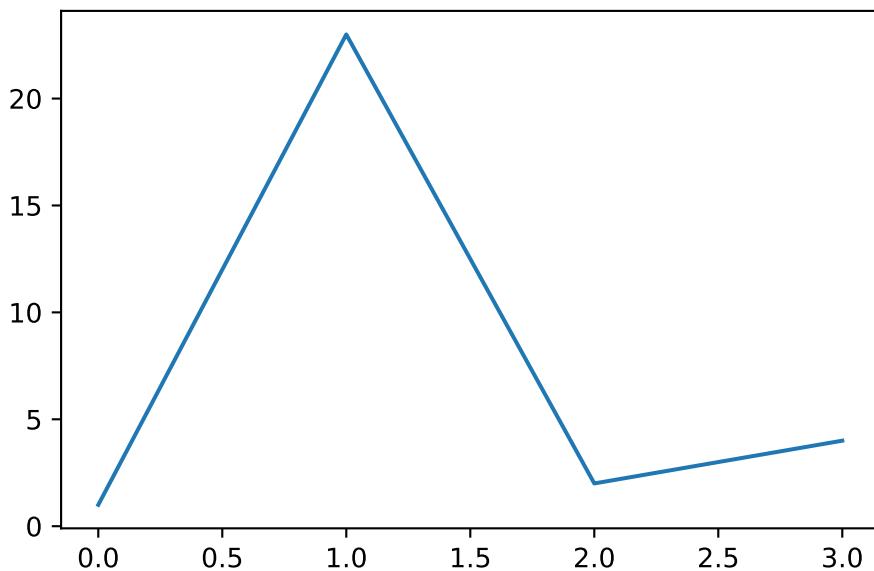


Figure 5.1: Simple Plot

In some cases, you may want to include a code-block that is not executed when the book is compiled. Use the `eval: false` option for this.

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

Figures can also be generated that do not show the code by using the option for `code-fold: true`.

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

r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
fig, ax = plt.subplots(
    subplot_kw = {'projection': 'polar'}
)
ax.plot(theta, r)
ax.set_rticks([0.5, 1, 1.5, 2])
ax.grid(True)
plt.show()
```

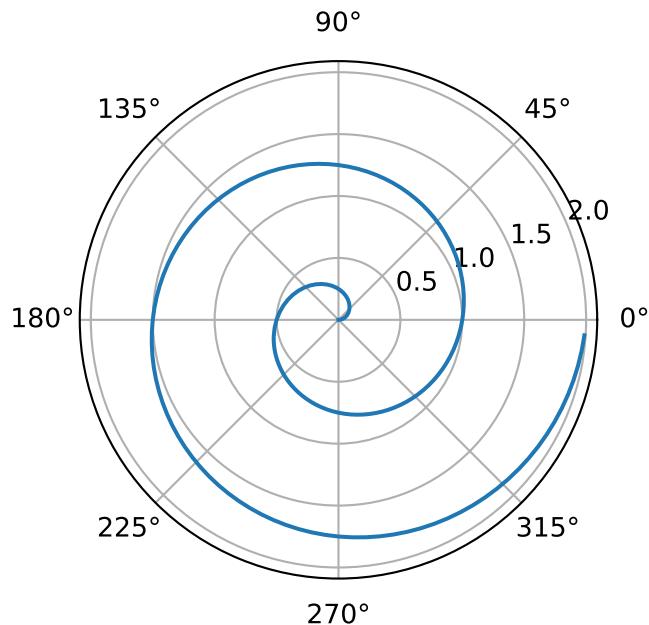


Figure 5.2: A spiral on a polar axis

Here is an example equation.

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (5.1)$$

5.4.1 Embedding Figures

You can also embed figures from other notebooks in the repo as shown in the following embed example.

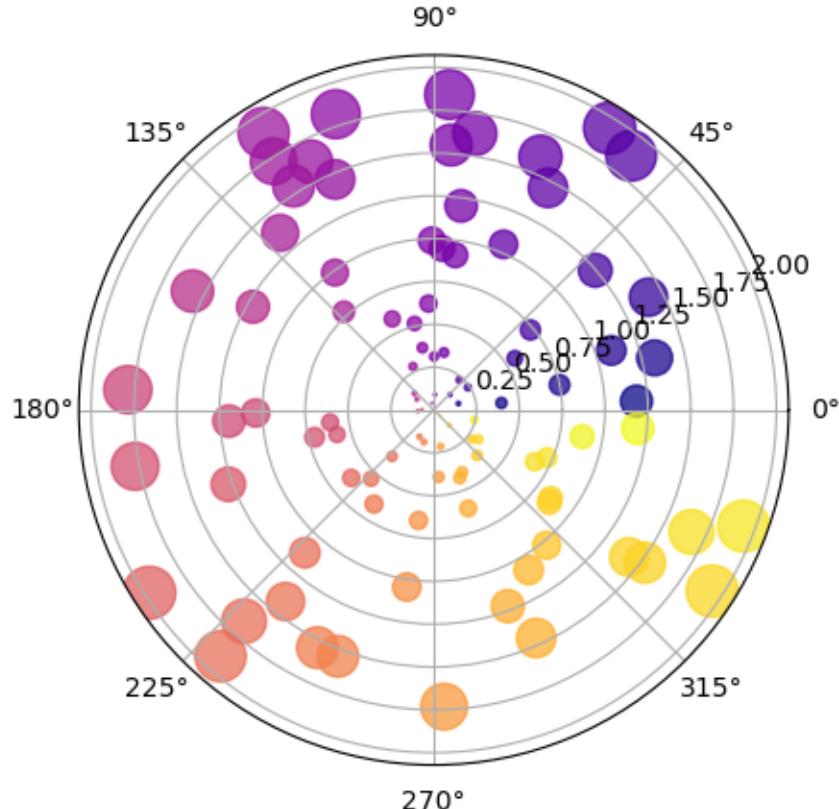


Figure 5.3: Polar plot of circles of random areas at random coords

When embedding notebooks, please store the .ipynb file in the notebook directory. Include the chapter in the name of your file. For example, `chapter4_example_u-net.ipynb`. This is how we will handle chapter- or example-specific environments. We will host notebooks on Google Colab so that any required packages for the code—but not for rendering the book at large—will be installed there. That way, we will not need to handle a global environment across the book.

5.5 Quarto has additional features.

You can learn more about markdown options and additional Quarto features in the [Quarto documentation](#). One example that you might find interesting is the option to include callouts in your text. These callouts can be used to highlight potential pitfalls or provide additional optional exercises that the reader might find helpful. Below are examples of the types of callouts available in Quarto.

Note

Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`. They can default to open (like this example) or collapsed (example below).

Tip

These could be good for extra material or exercises.

Caution

There are caveats when applying these tools. Expand the code below to learn more.

```
r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
```

Warning

Be careful to avoid hallucinations.

Important

This is key information.

6 Extending Your Hardware With AI

Image Restoration and Related Tools

Under your first header, include a brief introduction to your chapter.

Starting prompt for this chapter: This image restoration focused chapter motivates how AI can improve the image quality beyond hardware limitations. This chapter should include image restoration (denoising) and sensor-less AO (deep-learning-based AO); other topics can be included at the author's discretion.

Notes from the authors' meeting: This chapter can start with a broader overview of the topic, which will reference existing reviews on the topic. Then the chapter will focus on a denoising tutorial, which will include discussions of potential pitfalls and best practices.

6.1 Include section headers as appropriate

Use markdown heading level two for section headers. You can use standard markdown formatting, for example *emphasize the end of this sentence*.

This is a new paragraph with more text. Your paragraphs can cross reference other items, such as Figure 11.1. Use `fig` to reference figures, and `eq` to reference equations, such as Equation 11.1.

6.1.1 Sub-subsection headers are also available

To make your sections cross reference-able throughout the book, include a section reference, as shown in the header for Section 11.4.

6.2 Bibliography and Citations

To cite a research article, add it to references.bib and then refer to the citation key. For example, reference Stringer et al. [62] refers to CellPose and reference Chamier et al. [14] refers to ZeroCostDL4Mic.

6.3 Adding to the Glossary

We are using the extension [Quarto-glossary](#) to create a glossary for this book. To add a definition, edit the glossary.yml file. To reference the glossary, enclose the word as in these examples: LLMs suffer from . It is important to understand the underlying to interpret your results. Clicking on the word will reveal its definition. The complete glossary for the book will be listed in the [Glossary](#).

6.4 Code and Equations

This is an example of including a python snippet that generates a figure

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

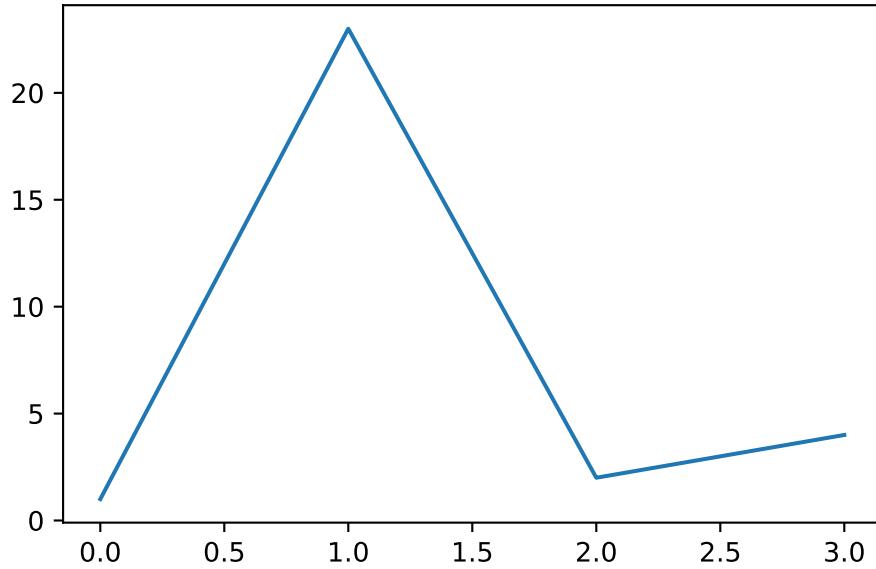


Figure 6.1: Simple Plot

In some cases, you may want to include a code-block that is not executed when the book is compiled. Use the `eval: false` option for this.

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

Figures can also be generated that do not show the code by using the option for `code-fold: true`.

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

r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
fig, ax = plt.subplots(
    subplot_kw = {'projection': 'polar'}
)
ax.plot(theta, r)
ax.set_rticks([0.5, 1, 1.5, 2])
ax.grid(True)
plt.show()
```

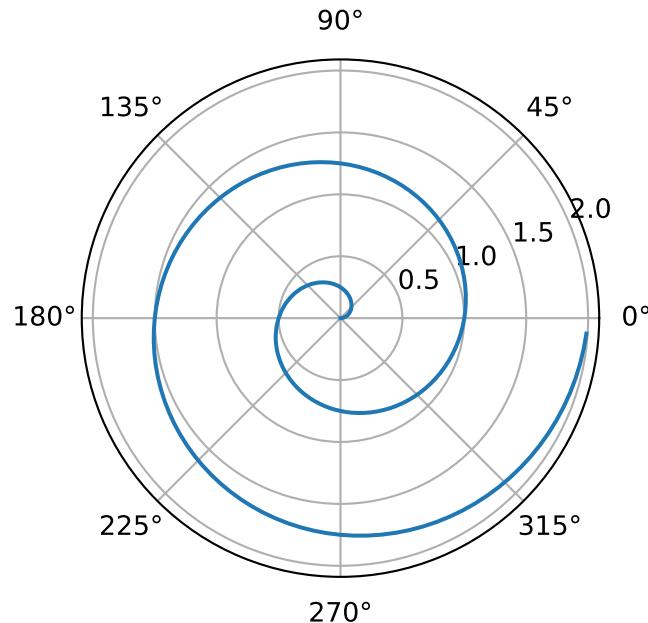


Figure 6.2: A spiral on a polar axis

Here is an example equation.

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (6.1)$$

6.4.1 Embedding Figures

You can also embed figures from other notebooks in the repo as shown in the following embed example.

When embedding notebooks, please store the .ipynb file in the notebook directory. Include the chapter in the name of your file. For example, `chapter4_example_u-net.ipynb`. This is how we will handle chapter- or example-specific environments. We will host notebooks on Google Colab so that any required packages for the code—but not for rendering the book at large—will be installed there. That way, we will not need to handle a global environment across the book.

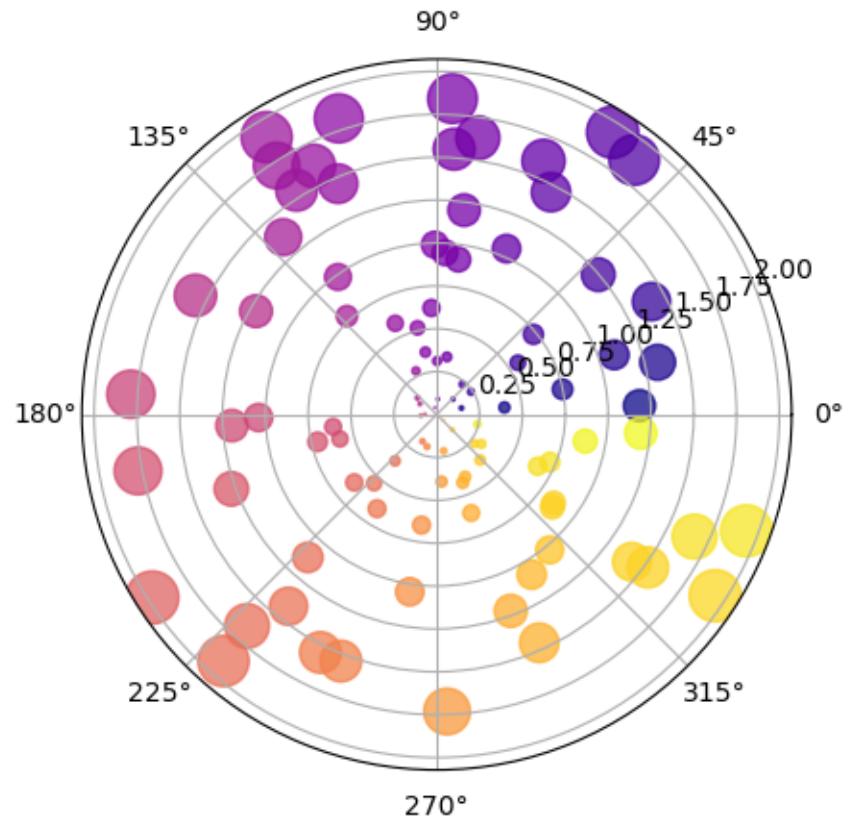


Figure 6.3: Polar plot of circles of random areas at random coords

6.5 Quarto has additional features.

You can learn more about markdown options and additional Quarto features in the [Quarto documentation](#). One example that you might find interesting is the option to include callouts in your text. These callouts can be used to highlight potential pitfalls or provide additional optional exercises that the reader might find helpful. Below are examples of the types of callouts available in Quarto.

Note

Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`. They can default to open (like this example) or collapsed (example below).

Tip

These could be good for extra material or exercises.

Caution

There are caveats when applying these tools. Expand the code below to learn more.

```
r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
```

Warning

Be careful to avoid hallucinations.

Important

This is key information.

7 Adding AI to Your Hardware

An Introduction to Smart Microscopy

- Introduction
 - Define and motivate the “problem”
 - * What is a Biological “Event”?
 - Why is it important to study these?
 - How is it distinct from an “Object”?
- Setting the bounds
 - What kinds of signals are we looking to extract events from?
 - What spatiotemporal scales are relevant?
 - What types of microscopes/imaging assays are in the scope of this chapter?
 - * Limiting the scope
- Brief history/evolution of automated event/object detection in general
 - Offer some background to current methods
 - * Including classical up to SVM
 - e.g micropilot and limitations
 - Case Study #1 : CellProfiler
 - Case Study #2 : Micropilot
- The need for new methods
 - Deep learning based and ML approaches
 - * Where the algorithm learns what is important
 - * What are the advantages of these methods?
 - What has been done recently in the literature?
 - * What is required to label, train and implement these methods?
 - Special considerations for event detection modes!
 - Specifics of event detection in contrast the other ML tasks
- Real-time vs a posteriori inference

- Challenges, opportunities
- Event detection as the first step in the microscopy workflow
- Limitations and notes of caution related to inference
 - “Trusting the algorithms”
- Conclusion

Part III

Image Analysis

8 Chapter 8: How do you select and find a tool?

8.1 Introduction

We are now transitioning away from a discussion of hardware and into a discussion of software. And, more specifically, in this chapter we focus on image analysis, particularly the human thought processes and decisions needed for selecting what software to use and figuring out how to appropriately/best use it. Amazing technological developments have occurred since the invention of microscopy almost half a millennium ago; Our current relationship to biological microscopy is profoundly shaped by the development of both quantitative microscopy and artificial intelligence in the mid-twentieth century. However, underlying the actual use of any technological development is a human (in our case, likely a biologist) selecting a specific technology to solve a specific problem.

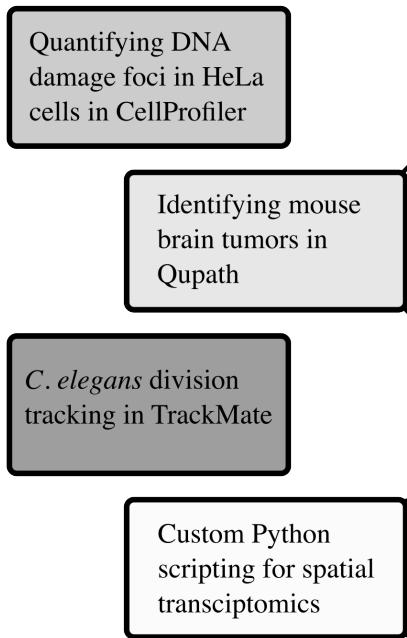
In this chapter, we answer the questions of how do you find new-to-you AI models and how do you assess whether a new-to-you model or tool will meet your needs? In this chapter, we first help you assess your needs. We then introduce a way of categorizing tools and help you use your needs assessment to select the right tool category. We suggest several places where you can find AI models for bioimage analysis and describe how to assess how well tools in those locations meet your needs. Finally, we show a couple of case studies that fulfill different requirements.

8.2 Assessing your requirements for a tool (in a perfect world)

Before finding and selecting an AI tool, you first need to decide on the type of tool you need. Your tool selection will be influenced by a number of factors and you should first honestly answer the questions that follow.

Figure 1: Workflows contain modular components

Image analysis workflows are not disparate monoliths...



...but are instead composed of many interchangeable and/or related components.

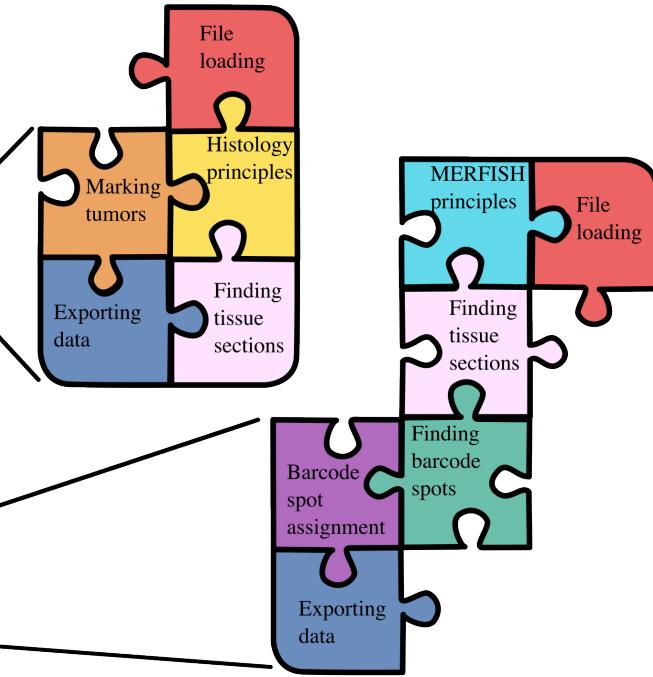


Figure 1. When thinking of your workflow and how you might incorporate an AI tool, consider that a workflow (left of image) typically consists of multiple, modular subcomponents (right of image). You may be able to replace your whole workflow with an appropriate AI tool or you may be able to replace only a subcomponent while still keeping many of the original workflow subcomponents the same. Replacing a single subcomponent with an AI tool may force you to change surrounding subcomponents in your workflow - either with new tools or new configurations of existing tools - as the AI tool may require different inputs and produce different outputs.

Though you may have built a workflow across multiple softwares, it is worth considering whether a single software (or a reduced number of softwares) can run your workflow and part of this consideration includes assessing how well your desired AI tool can run from within other software. Some examples of softwares that can be used to run other softwares include CellProfiler Stirling et al. [61], Fiji Schindelin et al. [53], Bilayers Shah, Gogoberidze, and Cimini [58], ZeroCostDL4Mic Chamier et al. [14] or DL4MicEverywhere Hidalgo-Cenalmor et al. [27], and Jupyter notebooks Kluyver et al. [30].

Figure adapted from Haase et al. [24].

8.2.1 Questions about your overall workflow:

- What is your desired output?**

You must be able to concisely and specifically state what you would like output by your workflow. Do you need to segment objects? If so, do you need [semantic](#), [instance](#), or [panoptic segmentation](#) (Box 1)? (see Chapter 4 and Chapter 9 for more information) Do you need measurements and if so should they be made on individual objects or on images?

- How will you assess your outputs?**

Will your assessment be qualitative or quantitative? Do you have ground truth? Do you have the expertise to assess output quality? (see Chapter 10 for more information)

- What is your desired “quality” of your outputs?**

Are you expecting outputs that approach ground truth or will “quick and dirty” be enough for you?

- What does your ideal workflow look like?**

Do you need all of the outputs to be made in a single software or are you comfortable using multiple softwares? If your images require preprocessing, does that need to happen in the same step? The same software?

Ground Truth

Ground truth can mean many things. The Broad Bioimage Benchmark Collection (BBBC) Ljosa, Sokolnicki, and Carpenter [39] provides a nice example of this. In the BBBC, all datasets include ground truth, but that ground truth can be one or many of counts, foreground/background, outlines of objects, biological labels, location, or bounding boxes.

8.2.2 Questions about your resources: time, expertise, and compute:

- How much time are you able to put into the task?**

An honest assessment at the beginning of any project about the time you are willing to invest is critical. If you are in a rush, you’ll probably want to select a method that is already familiar or best matches your existing competencies.

- What is your priority?**

Determining what your priority is goes hand in hand with assessing the amount of time you can put into your task. Perhaps you’re in a time crunch and speed is the most important consideration for you. Perhaps you have a specific [quality control \(QC\)](#) metric and you need the tool that will give you the outputs that optimize this metric. Perhaps you’ve always wanted to gain experience with a particular class of tool, so you want to

figure out if this is the right use case for trying it.

- **What is your level of computational domain expertise and comfort?**

If you don't have a high level of computational comfort, do you have the time and motivation to expand your skillset by building new skills outside of your comfort zone? Do you have a computational collaborator and how much time, in either teaching you or in handling the data themselves, are they able to contribute? Computation domain expertise has two critical, but separable, components. The first is the ability to understand what you are doing thoroughly enough that you can design, analyze, and interpret your experiment. The other component is the ability to comfortably interact with software as a computationalist might (e.g. are you comfortable working from the command line (which is text only) or would you prefer a GUI (graphical user interface), where you can point and click).

- **How do you like to learn things?**

Are you a person who likes to jump in and just turn all the knobs and buttons to see what they do? Do you love to cozy up with a good written manual at the end of the day? Do video walkthroughs bring you joy or frustration? How important is it to you that you be able to ask the tool creator questions directly? The type of documentation and instructional materials as well as their depth can vary greatly between tools.

- **What is your level of biological domain expertise?**

Are you confident that you fully understand the task so you can assess how well your AI model is performing? Do you have a biologist collaborator and how much time are they able to contribute to designing the experiment and/or analyzing data? Do you understand what controls you will need and/or corrections (such as single-color controls and/or measured flat-field corrections) you will need to make to be able to interpret your outputs?

- **What access do you have to compute resources?**

Do you need to be able to run everything on your laptop or do you have ready access to a high performance cluster or scalable cloud compute? Do you have access to GPUs? Different tools have different compute requirements, especially as you scale them, and those requirements don't always scale linearly with data size.

8.2.3 Questions about your input data:

- **How difficult are your images?**

Perfect, clean data is the ideal input to any analysis. But that's not always the regime we're in. There are many sources of "difficult". A couple examples and corresponding questions are below.

- Do your images have debris or other technical artifacts such as out-of-focus regions?
Do those artifacts need to be identified and removed? If identified, should the whole

image be dropped or do you need to keep non-artifact areas of the image?

- Do you have **metadata** that needs to be associated with your images? Is that metadata organized? How is that metadata organized (e.g. in the file name, in a .csv file, in a picture of a handwritten lab notebook) and does it play nicely with the tool you would like to use?
- Are your images in a friendly format? Are they in a proprietary file format? Are they in a file format that allows access to chunks?

- **How big is your data?**

The larger data is, the harder it can be to work with. If your data is large, what dimensionality is big? e.g. many images, large individual file sizes, many channels

- **Do you need outputs in multiple dimensions?**

If you have z-planes, do you need 3D objects or can each plane be handled separately? If you have multiple timepoints, do you need objects tracked across timepoints or can each timepoint be handled separately?

- **Do your images require preprocessing?**

There are many different reasons that images might require preprocessing and some of those reasons may be a way to overcome concerns/technical challenges raised in other questions above. Some examples of preprocessing include stitching or cropping of images, denoising, background subtraction, or flat field correction.

Box 1: Segmentation methods

In computer vision, there are several discrete kinds of segmentation, although in the field of bioimaging we often refer to all kinds of segmentation under the single blanket term of “segmentation”. **Semantic segmentation** divides an image into classes. Ilastik is an example of a popular image analysis software that performs semantic segmentation. **Instance segmentation** detects individual, specific objects within an image. CellProfiler is an example of a popular image analysis software that is most commonly used for instance segmentation. **Panoptic segmentation** is a combination of semantic segmentation and instance segmentation that separates an image into regions while also detecting individual object instances within those regions. Deep learning can be used for semantic, instance, or panoptic segmentation. Most classic image analysis methods are built around instance segmentation for cell-based images, though semantic segmentation is not uncommon for tissue/histology images.

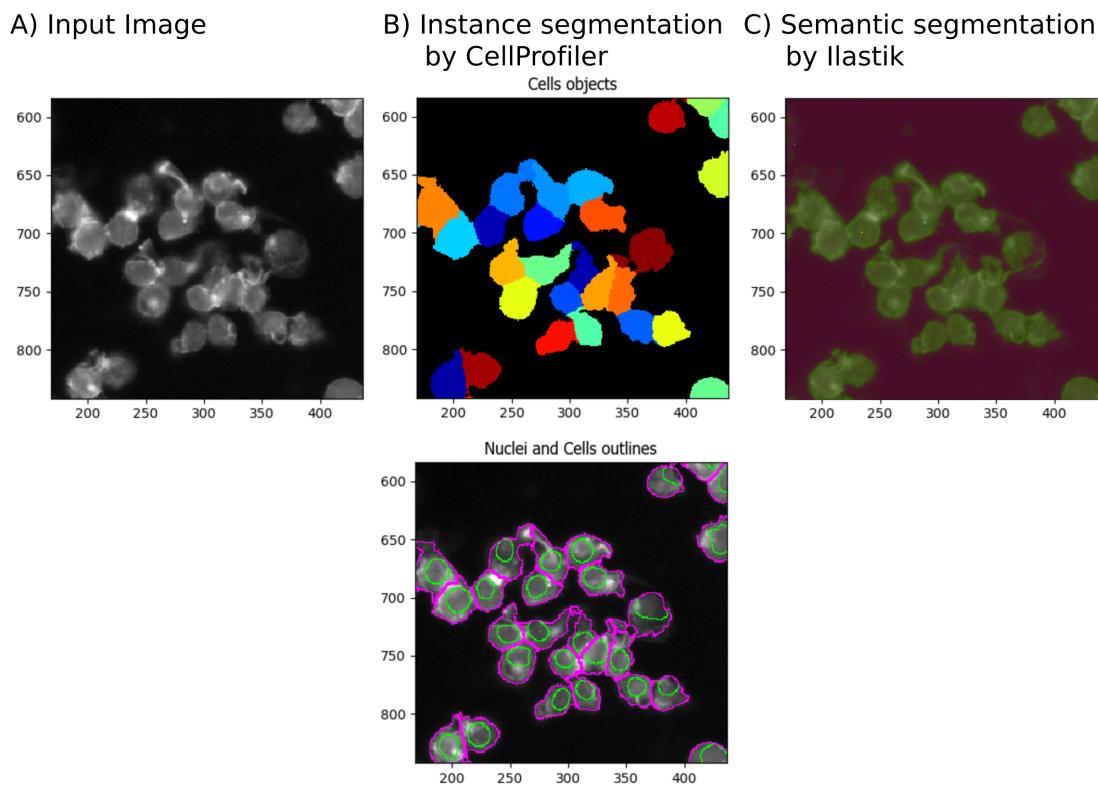


Figure 8.1: Segmentation comparison of CellProfiler and Ilastik

Example of different types of segmentations produced between CellProfiler and Ilastik. A) Input image of Drosophila Kc167 cells, provided in the Example pipeline packaged with CellProfiler. B) CellProfiler uses classic image processing to create instance segmentations. The individual objects identified are shown as individually colored masks in the upper panel. The lower image is the input image overlaid with nuclei objects outlined in green and cell objects outlined in pink.

C) Ilastik uses pixel-based machine learning to create semantic segmentation. The image is the input image with green shaded areas in the class “Cells” and red shaded areas in the class “Background”.

8.3 Assessing your requirements for a tool (in the real world)

In several previous surveys we have run, scientists have reported that they generally don’t actually follow any thoughtful, rigorous process for deciding what tool they will use for a particular process. Instead, they sit down with a thing they know how to use and hope they

can make it follow their use case Jamali et al. [29] Sivagurunathan et al. [60]. If they don't already have familiarity with a tool that is appropriate for their task, they ask their labmates or maybe do a quick internet search. So know that if you found the section above overwhelming, you are not alone.

If you already have familiarity with a tool and it fits your newest use case (especially if your newest use case isn't *very* different from a use case where you've previously successfully used it) go ahead and stick with that tool. "If it ain't broke, don't fix it" is a common adage for a reason. But even for existing use cases, it's worth actively contemplating your priorities every-so-often - maybe your GPU cluster is filling up these days and is harder to access, or after a new operating system upgrade a tool that was working great is now randomly crashing, or maybe there simply is something new (or new-to-you) available that will make your life easier. In our experience, the best way to be sure you're best serving your scientific needs is simply to step back and consider your big picture goals and whether your tool choices are serving those goals with some regularity.

Once you have conducted a thorough needs assessment, it should *theoretically* be simple to determine if a given tool or model is the right one for you. Unfortunately, no tool is likely to perfectly fit every single need that you have! You will likely need to break your needs into "must haves" and "nice-to-haves", and rank tools accordingly. Make sure you consider not just how well the tool will perform your specific task, but also how well it will fit in your workflow (Figure 1)

8.4 How do I decide when my workflow is “good enough”?

Deciding when a workflow is “good enough” can be really hard! We generally factor in two major considerations when deciding what quality level we are aiming to achieve: Is this a pilot or final data? and How big is the [effect size](#) we’re looking for?

We generally hold pilot data to a lower quality standard than final or production data. We are looking for a proof of concept that our assay/method can work but understand that there are likely to be refinements and improvements at many points in the workflow that will increase our final quality. Additionally, during the piloting phase, there are likely to be changes introduced that change the inputs to our workflow so time spent beyond a certain point of workflow refinement is lost with the retuning/adjustments required for the next pilot batch.

Phenotypic effect size has a large impact on necessary quality level. Simple, easily distinguished phenotypes (e.g. GFP is either in the nucleus or cytoplasm) may reach statistical significance with many fewer individual measurements and using much simple/cruder regions of interest (e.g. cell bounding boxes instead of careful cell margin segmentations) than more subtle phenotypes.

Research progress is rarely linear

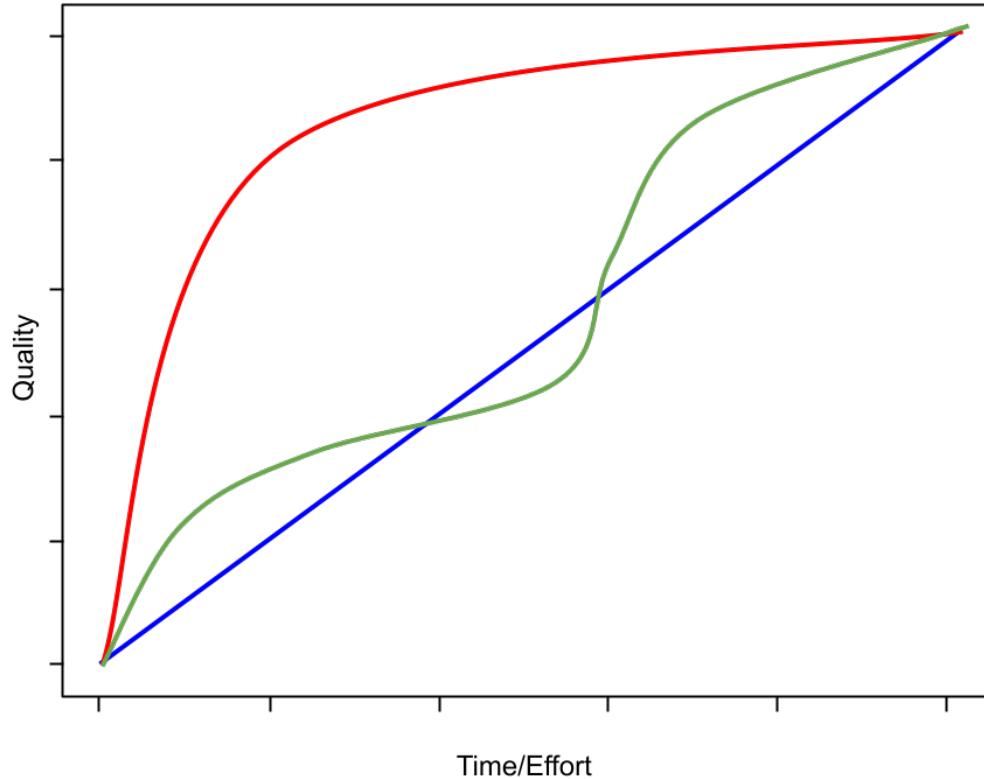


Figure 2. As scientists who care about the quality of our work, it's tempting to always try to maximize the accuracy of our analysis. But it's worth considering that "accuracy" vs "time spent" is often an asymptotic curve - you can keep pushing the accuracy higher and higher, but the gains may be marginal past a particular point. Time and effort spent on a project can have very different relationships with the output quality. Though one might hope to be in a regime where the amount of time spent is linearly correlated with the quality of a project's output (blue line) so that it is clear exactly how much time must be put in to get a desired product, research rarely proceeds in such a fashion. Instead, it progresses in unpredictable fits and starts (green line) and/or it is in a state of diminishing returns (red line) where the closer you get to "perfect", the longer it takes to improve the quality.

In Cimini [17] we detail and provide examples for a few key heuristics that we think about when deciding if a workflow is "good enough". Those heuristics are:

1. How close to accurate do I need to be to assess the % or fold change I expect to see in this experiment?

2. How close to 100% accurate is it possible to get with my current workflow?
3. How important is this aspect of my experiment to my overall hypothesis?
4. How wrong is my current pipeline output?
5. What else could I do (e.g. develop a different approach/workflow, learn a new tool, generate new input data) in the time it will take me to make my pipeline maximally accurate?

8.5 Choosing the kind of tool to use

Let's consider the use case of instance segmentation (see Box 1), which is often a critical step in image analysis workflows. We have defined 6 categories of segmentation methods, each of which has its valid use cases. Ordered by increasing levels of computation comfort required they are:

- [Manual annotation](#)
- Classical image processing
- Pixel based machine learning
- Pretrained deep learning models
- Finetuned deep learning models
- From-scratch deep learning models

We will not cover from-scratch deep learning models in this chapter. They are covered in Chapter [9](#).

It is worth noting that many of the tool suggestions below are not confined to a single “class”. Many tools listed in the non-deep learning categories allow you to run pre-trained deep learning models inside them (e.g. CellProfiler has a RunCellPose module, ImageJ and QuPath also contain ability to run [pixel classifiers](#)). All deep learning models *can* be re-tuned with some effort (they may just not have a friendly interface for doing so).

Method category	Suggested light microscopy tools	Time required	Best if your priorities include	Biological level required	Ability to handle noise and variability	Largest drawback
Manual Annotation: Hand-drawing in boundaries of all objects	ImageJ Schindelin et al. [53] Schindelin et al. [55], Napari <i>napari: a multi-dimensional image viewer for Python</i> [43], see also this forum thread for 3D annotation	Highest	High accuracy/ outputs approaching ground truth	High	High	Inter- and even intra-operator manual annotations (e.g. after time or image rotation) variability. Irreproducibility. Can not be automated.

Method category	Suggested light microscopy tools	Time required	Best if your priorities include	Biological level required	Ability to handle noise and variability	Largest drawback
Classical Image Processing: Combining mathematically well-defined operations such as thresholding and watershedding to create objects	CellProfiler Stirling et al. [61], QuPath Bankhead et al. [8], Icy Chaumont et al. [15]	Medium	Needing to have a good sense of what the failure cases will be (e.g. when cells are too big, too small, too crowded, etc); needing to run in minimal computational footprints	High	Initially low, can be improved with experience	Most performant on very bright objects on dark backgrounds

Method category	Suggested light microscopy tools	Time required	Best if your priorities include	Biological level required	Ability to handle noise and variability	Largest drawback
Pixel based machine learning: Using properties of individual pixels to classify each pixel into belonging to a given set of “classes” (e.g. “true signal”, “autofluorescence”, and “background”)	Ilastik Berg et al. [11], Labkit Arzt et al. Using [7]	Low to Medium	Identifying well-separated objects that are not bright-on-dark or are not the <i>only</i> bright-on-dark objects present	High	High if properly represented in the training set	Must be followed by Classical Image Processing if individual object masks are required

Method category	Suggested light microscopy tools	Time required	Best if your priorities include	Biological level required	Ability to handle noise and variability	Largest drawback
Pre-trained deep learning models: Using a set of learned transformations (created by the developer) to define the positions and boundaries of objects	Stardist Schmidt et al. [56], Instanseg Goldsborough et al. [22]	Low	Maximum likelihood for good quality with minimal human time required	High	High if properly represented in the training set (which you may not know)	Unlikely to work if data similar to your data is not present in the training set. Can be hard to package into workflows.
Fine tuned deep learning models: Updating or adding to a pretrained deep learning model with your data	Cellpose Pachitariu and Stringer [47], μ SAM Archit et al. [4]	Medium	Maximum likelihood for very good quality with less human time than manual annotation	High	High if properly represented in the training set	Most computationally intensive. Many available models are not provided in a non-computationalist friendly format. Can be hard to package into workflows.

You may have noticed that all methods are listed as requiring a high level of biological expertise. It can be tempting for researchers without deep biological expertise to assume that they could just use a pretrained or fine tuned deep learning model and call it a day. However, we would caution that, while they may require less of a biologist's time to tune at a first pass, it is still essential to involve a biologist with subject matter expertise to verify that the models are producing results that match known biology.

8.6 Finding tools and models

8.6.1 Where to look for tools and models

There are many places that one can look to find new (or new-to-them) tools and models. A few suggestions are as follows:

- BioImage Model Zoo Ouyang et al. [44] is a community-driven AI model repository that offers a variety of pretrained AI models. All models are described by a common Model Resource Description File Specification which simplifies using models in the BioImage Model Zoo with partner software such as ilastik Berg et al. [11], ImJoy Ouyang et al. [45], Fiji Schindelin et al. [53], deepImageJ Gómez-de-Mariscal et al. [23] and ZeroCostDL4Mic Chamier et al. [14].
- Bioimage Informatics Index (Biii) Zhang et al. [63] is a community-curated index that includes software and helps researchers find tools whether their search is problem-based (e.g. “find nuclei in cells”), method-based (e.g. “active contour-based segmentation”), or tool-based (e.g. “CellProfiler”).
- The Scientific Community Image Forum (forum.image.sc) Rueden et al. [51] is a platform where users can pose bioimage software-related questions and receive feedback from a broad user community and maintainers from >75 partner softwares.
- Academic literature can include biology-focused research where users describe applications or computer science research where the tools and models themselves are more likely to be described. Keeping an eye on preprint servers like arXiv (particularly in Computing Research Repository or Quantitative Biology sections) and bioRxiv can help you find the most cutting-edge developments.

Bioimage analysts

There is a whole discipline of people who specialize in figuring out workflows for analyzing microscopy images - their official titles often vary, but many think of themselves as *bioimage analysts*. If you want to find a local bioimage analyst, or even become one, you can check out the [Global Bioimage Analysts' Society \(GloBIAS\)](#), which holds events,

coordinates trainings (including free help sessions), and hosts a database of bioimage analysts and bioimage analysis trainers.

8.7 Case Studies

8.7.1 Case Study 1:

8.7.1.1 Introduction

Elena is an image analyst who uses high content microscopy as a method for exploring drug mechanism of action. She uses an automated microscope to acquire 9 fields of view in each well of a 384-well plate and just acquired a batch of 4 plates worth of images. She often performs the same assay so she has a series of pipelines that she has nicely tuned for her usual experimental parameters. Elena has a new collaborator that would like to perform a similar screen in a different cell line as the collaborator is interested in a different specific area of biology. Because this is the first time that Elena has performed this assay in a new cell line, she needs to carefully tune her existing pipelines so that she has high-quality segmentation of Nuclei and Cell objects.

8.7.1.2 Workflow

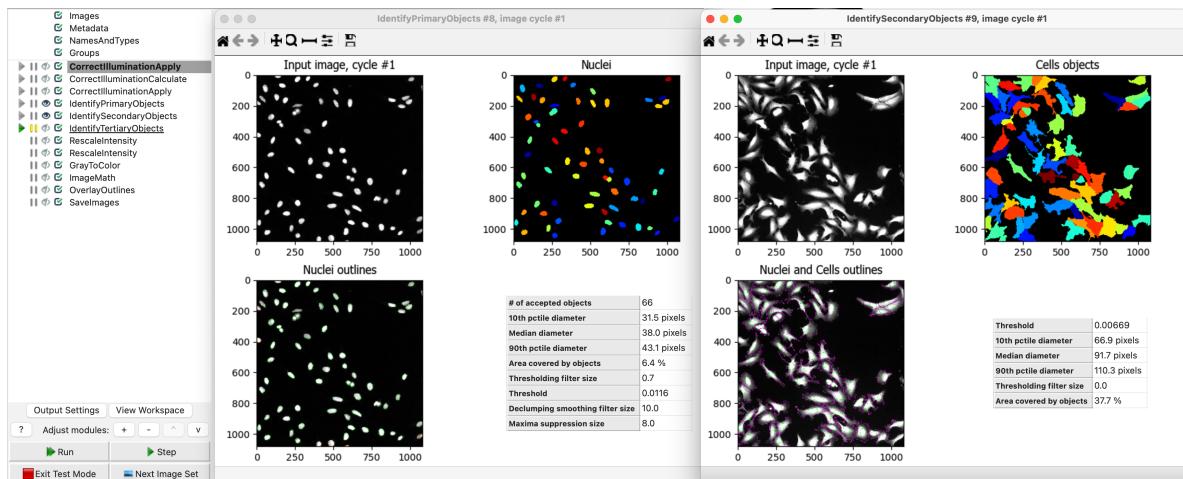


Figure 8.2: CellProfiler for nuclei and cell segmentation

Elena usually uses CellProfiler as an image analysis workflow tool, using classical image processing methods for object identification in her pipeline. Elena typically performs her

screens in A549, U2OS, or HeLa cells which are all considered easy to use for high-content microscopy because they are relatively consistent in size, relatively round, and can grow both sparsely and confluently while remaining in a monolayer. She's quite used to making parameter changes to the IdentifyPrimaryObjects and IdentifySecondaryObjects modules to fine-tune her nuclei and cell segmentations, respectively. The image above shows shows the modules in her CellProfiler pipeline (left) and the outputs of her IdentifyPrimaryObjects (center) and IdentifySecondaryObjects modules (right).

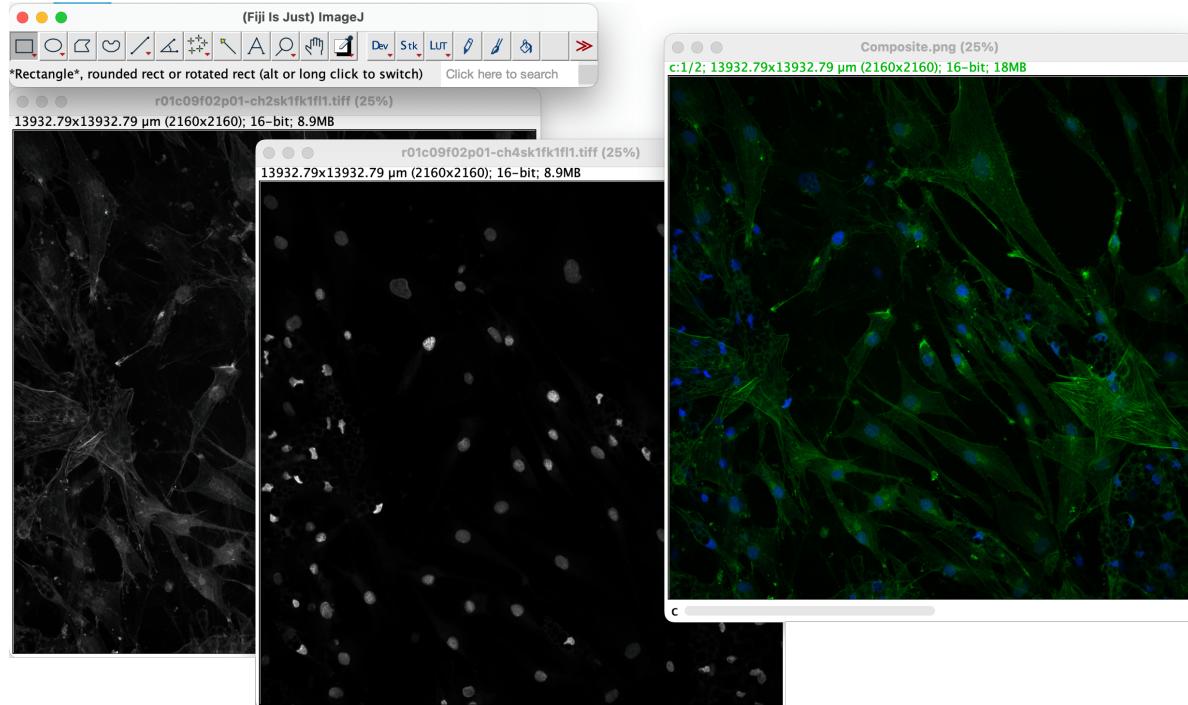


Figure 8.3: Fiji for compositing a new png image

Elena's collaborator has acquired images in an adipocyte cell line and they are morphologically quite different from her typical screening cell lines. After being unsatisfied with what she was able to tune using classical segmentation in her standard Cellprofiler pipeline, Elena decides to turn to an AI tool. She starts by opening Cellpose in its web console as she has heard from colleagues that it is both very good out of the box and allows a user to fine tune it for their own data. She initially struggles to load one of her own images into the web tool but notices that the web page says “upload one PNG or JPG <10 MB”. She also notices that the example images that are most similar to hers are in color, with a nuclear channel in blue and a cell marker in green. Elena’s raw images are in .tif format with a single grayscale channel per image so she uses Fiji to make a composite .png image. The image above shows her single-channel .tif images opened with FIJI (left, center) and the composite .png image that she created (right).

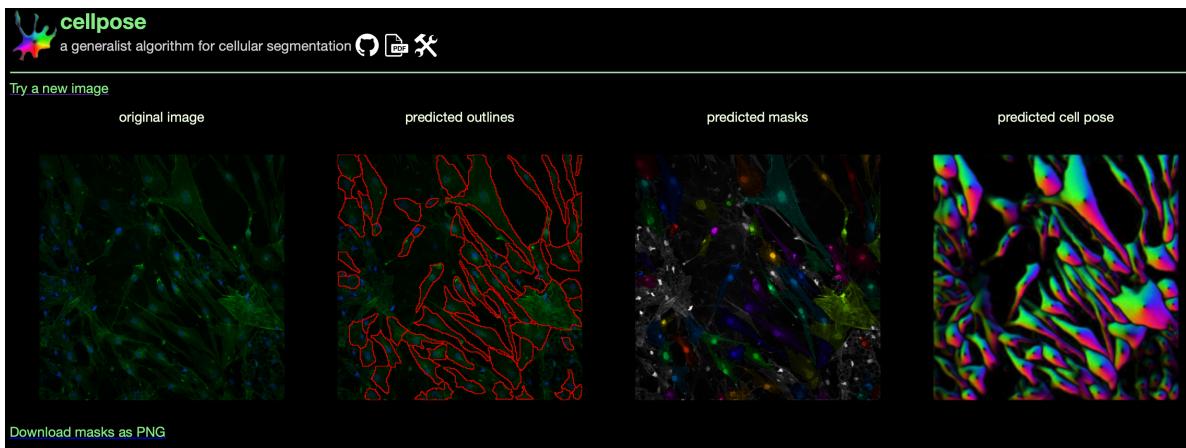


Figure 8.4: Cellpose web console

Elena loads her new composite .png into the Cellpose web console and sees that it has identified a number of reasonable cell objects but is not of the quality that she wants for use in her screen. However, Elena is excited to explore a new tool and has been actively building her computational skills so she decides to proceed with Cellpose and see if she can improve the outputs. The image above shows her example image loaded into the Cellpose web console (far left) along with the predicted object outlines (center left), masks (center right), and flows (far right).

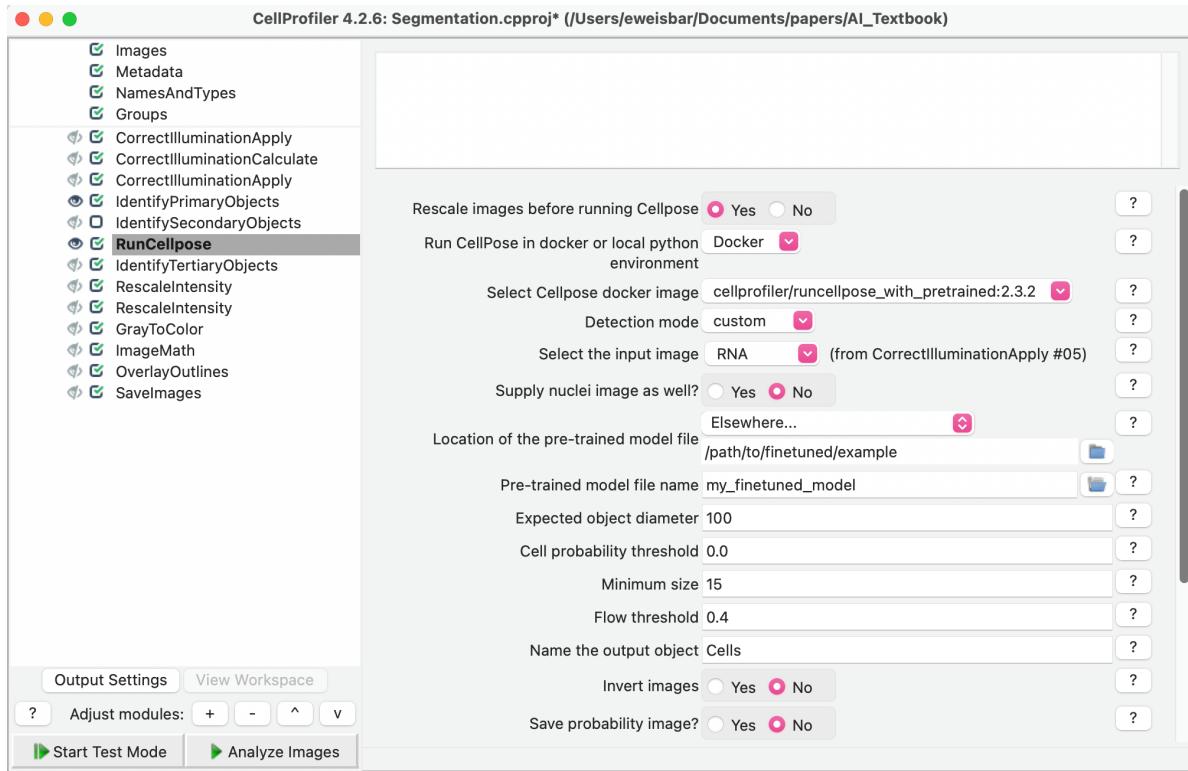


Figure 8.5: RunCellpose in CellProfiler

Elena takes advantage of the fact that Cellpose allows for fine-tuning of the pretrained deep learning models it provides. She re-trains Cellpose using a Colab notebook and sees that it now performs quite well on test images from her dataset. Because CellProfiler is also an image analysis workflow tool, Elena is able to insert a RunCellpose module in her original pipeline instead of the original secondary object identification module and keep the rest of her image handling and analysis the same while using the newer, better performing object identification that Cellpose provides. The image above shows her new CellProfiler pipeline with the RunCellpose module pointing to her custom fine-tuned Cellpose model.

8.7.1.3 Conclusion

This case study starts with a scientist trying to use her standard classical image processing methods for object identification but finding that she gets better results with a new challenging segmentation task by moving to an AI tool. It shows iterative improvements using the same tool in different ways and highlights some of the simple changes she needed to make to her overall workflow to use the new tool.

8.7.2 Case Study 2:

8.7.2.1 Introduction

Esteban is an image analyst, working with imaging data acquired by a collaborator. The collaborator studies zebrafish brains and sent Esteban fluorescent images taken in a large z-stack in live zebrafish. The zebrafish express a GFP-tagged protein in their microglia and the collaborators would like to be able to identify the microglia using an automated pipeline so that the researchers can ask biological questions such as where they are within the brain and whether they are close to other cell types (labeled in other fluorescent channels). The task is complicated because there is additional autofluorescence in the skin that is the same intensity as the microglia and they do not want skin autofluorescence identified. The images also have technical challenges because of the nature of their acquisition: because they are 3D live images, as you go deeper in the tissue the signal gets dimmer. This makes using normal [computer vision](#) segmentation quite hard.

8.7.2.2 Workflow

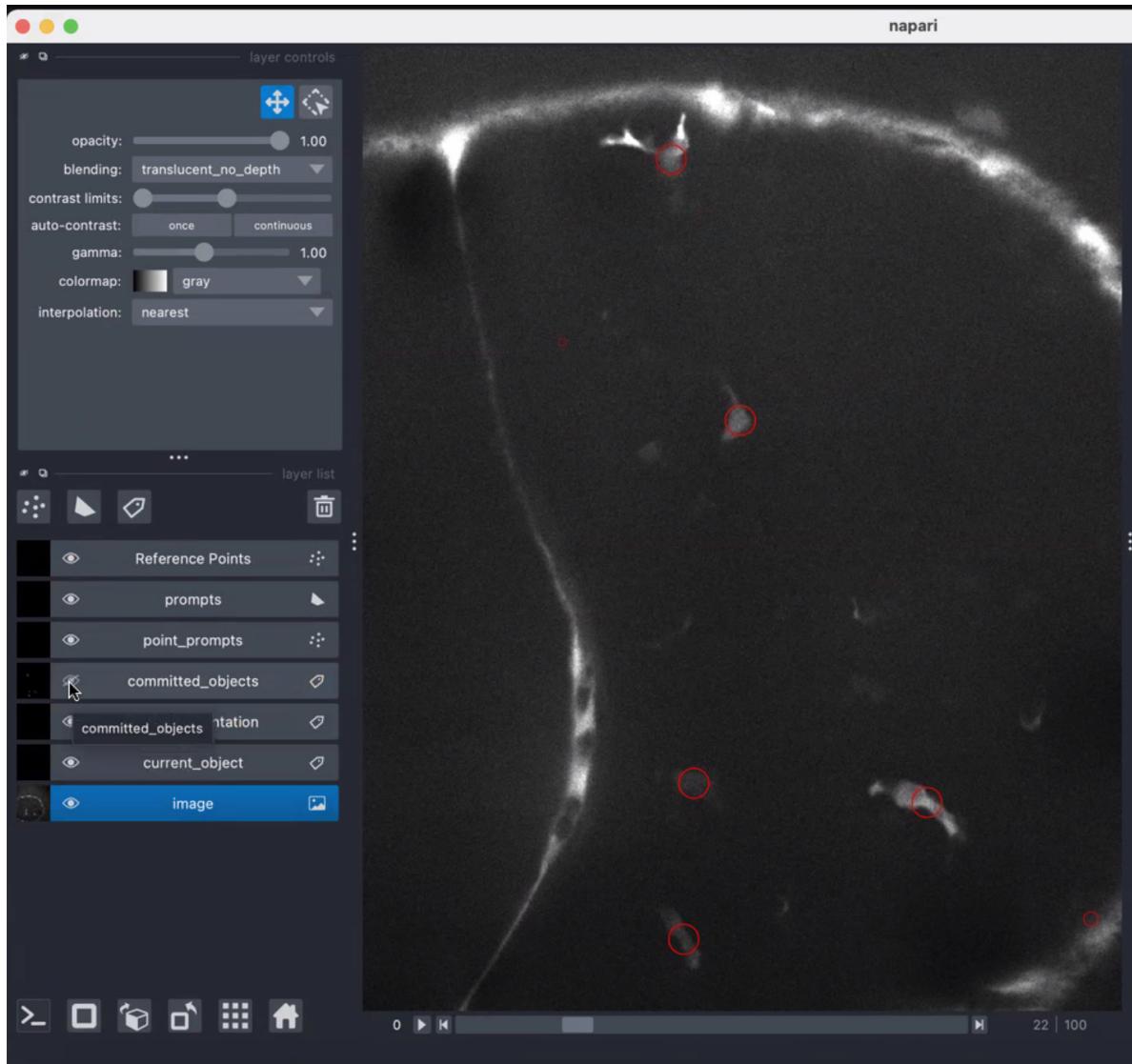


Figure 8.6: Point Annotations in napari

To help orient Esteban to the image data, he was given a spreadsheet of x,y,z coordinates of microglia that the collaborators manually annotated. Esteban started by opening the images and their point annotations in napari to orient himself to the data. The image above shows an example image opened in napari with red circles around the point annotations.

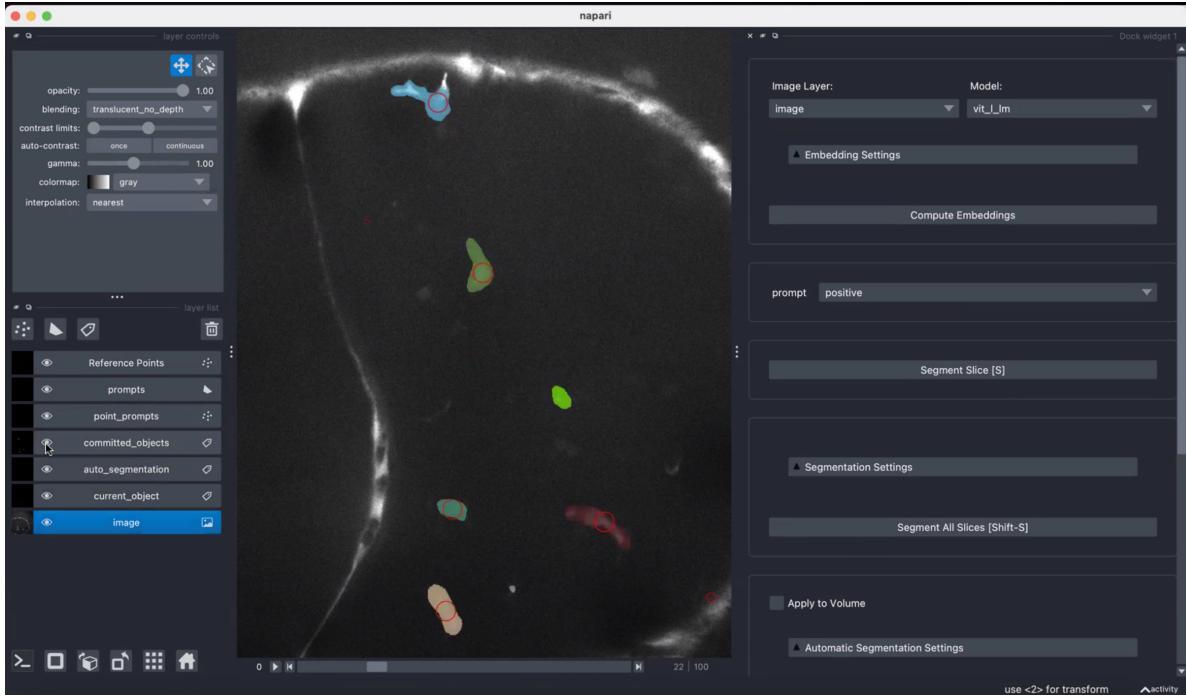


Figure 8.7: Guided Annotations in napari with SAM

Esteban then loaded μ SAM in napari to use as an annotation tool so that he could use the point annotations to make object masks. He tried using automatic segmentation but it didn't work on his images. Instead, he used μ SAM assisted segmentation, manually providing positive prompts, and got out reasonable segmentations of the microglia in the image. The image above is quite similar to the previous image but also has the μ SAM segmentations in color overlays.

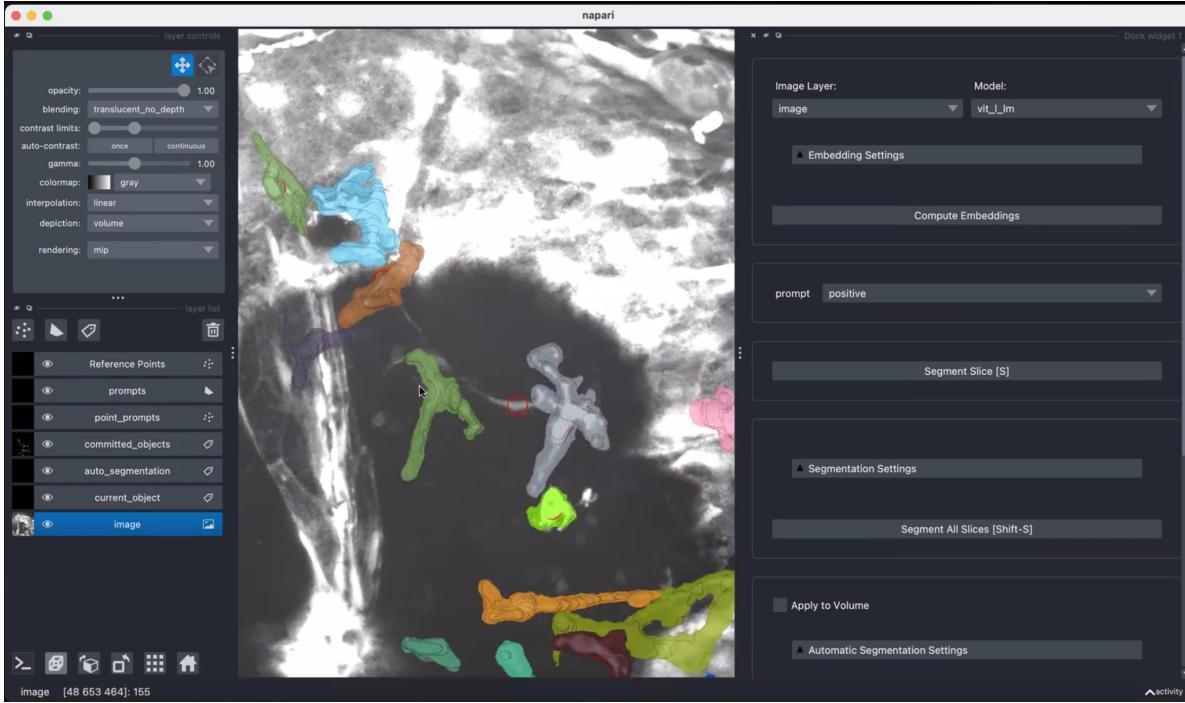


Figure 8.8: Visualizing 3D annotations in napari

Though Esteban has three dimensional data, the segmentations he is creating are 2D so he confirms that he is happy with his final masks by examining the image and masks in a 3D viewer. The above image shows a 3 dimensional view of his data that contains all of the image slices and annotations viewed together.

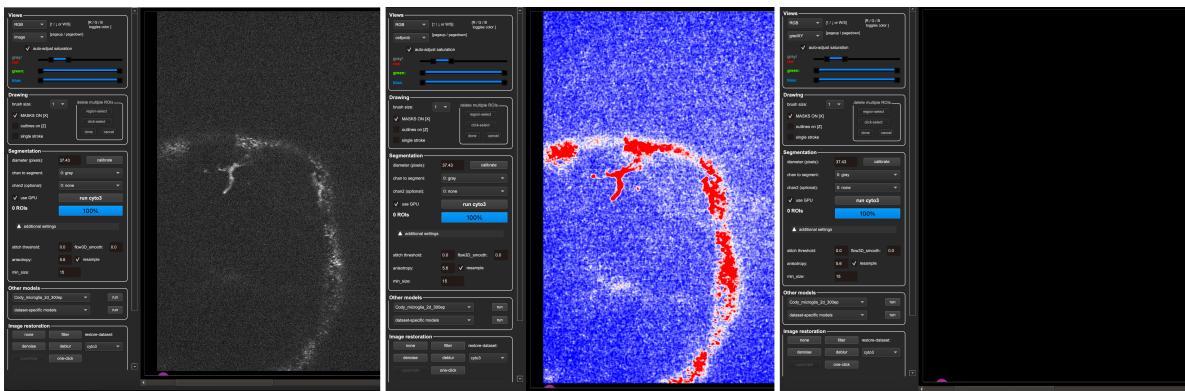


Figure 8.9: Cellpose predictions

Now that Esteban has both images and masks, he calls Cellpose in Python and retrains the cyto

model with his annotations. He runs the retrained model on a new image (left image) and finds that the cell probability prediction seems to identify microglia and autofluorescence (center image), but, unfortunately, the gradient prediction is not finding any cell objects (right image). Esteban's training images are quite sparse (the vast majority of the image is background) so the Cellpose model learned that it can be mostly correct by deciding that *everything* is background.

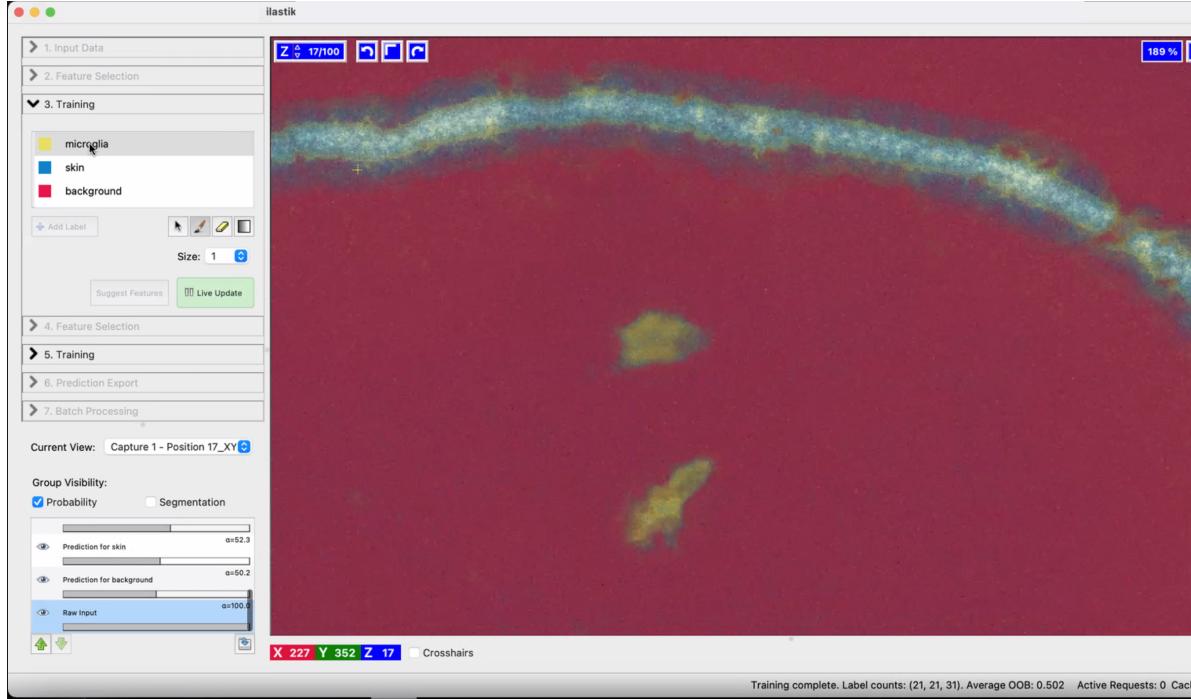


Figure 8.10: ilastik 3 class predictions

Since neither a pre-trained model (μ SAM) nor a fine-tuned model (Cellpose) worked on his images, Esteban decided to try pixel classification. He used ilastik, first providing annotations for microglia and background. The classifier worked quite well at distinguishing microglia from background, but classified the skin autofluorescence as microglia. Esteban built on this by using larger filters so that the classifier had more contextual information, training 3 classes (microglia, skin, and background) instead of 2, and adding in autocontext Kreshuk and Zhang [32] which improves classification by running it in multiple stages. Esteban's final 3-class pixel classifier performs well to identify microglia separately from skin across the images in his datasets. The above image shows a zoom-in of one of his images with the three classes overlaid - red for background pixels, blue for skin pixels, and yellow for microglia pixels.

8.7.2.3 Conclusion

This case study starts with a scientist trying multiple state-of-the-art deep learning segmentation tools for a specific, challenging segmentation task. He ultimately discovers that a machine learning pixel classifier works better for this task, illustrating that 1) “less-advanced” tools can perform better than deep learning tools and 2) biological expertise is always needed when inspecting any tool’s outputs.

8.8 Conclusion

As we shift our focus from hardware to software, it is common and understandable to feel overwhelmed - the bioimage analysis tool space is a constantly-changing array of best practices, and not all microscopists are highly computationally comfortable. This chapter therefore mostly talks about *how to make decisions*, as opposed to *use X tool for Y task*. Ultimately, the “right tool for your task” is the tool (or chain of tools) that allows you to be confident that your analysis has been performed **well enough**, and helps you understand the answer to your biological question and how confident you can be in your conclusion. Even if you try a tool and decide it isn’t the right solution for this question, this time still isn’t wasted - it simply enlarged the toolbox you can use for answering future questions. In the subsequent chapters, we will teach you about how to train models which could be important parts of your toolbox, but we suggest you keep our guidelines in mind as you finalize the decisions around when your model is “trained enough” and how you decide on your overall workflow.

9 How to Train and Use Deep Learning Models in Microscopy

Practical considerations through the lens of segmentation

9.1 Introduction

9.1.1 The challenge

The growing volume and complexity of image data necessitate increasingly advanced analytical tools. One example of challenging tasks is image segmentation, the process of identifying and delineating structures of interest within images. Segmentation can be particularly difficult and time-consuming when dealing with large, multidimensional datasets, such as 3D volumes or time-lapse sequences, where [manual annotation](#) becomes impractical. Machine learning (ML), especially deep learning (DL), can provide effective solutions to these challenges [25].

ML algorithms learn patterns from data to perform tasks such as [image classification](#) and segmentation. Traditional ML methods, like random forest classifiers, depend on manually defined image features to classify pixels [5, 6, 12]. In contrast, DL algorithms can automatically discover and extract relevant features directly from image data using multilayer neural networks, which eliminates the need for manual feature selection. DL techniques are widely applied in complex image analysis tasks, including segmentation, [object detection](#), feature extraction, denoising, and restoration [40, 48]. Due to their ability to automatically learn hierarchical features, DL methods usually achieve greater accuracy and efficiency than traditional ML techniques [33, 50].

Segmentation greatly benefits from ML and DL, as manual segmentation is extremely time-consuming and impractical for large datasets. This chapter offers practical guidance on preparing a segmentation project and emphasises effective DL applications to tackle these challenges. While we focus on segmentation as a case study, the principles, workflows, and considerations discussed here are broadly applicable to other image analysis tasks, such as classification or denoising. Readers interested in these areas can adapt the described approaches to their specific needs.

9.2 Preparing for your project

9.2.1 Defining your task and success criteria

Every image analysis project should begin by clearly defining the scientific question you wish to answer, along with the criteria by which success will be measured. These foundational decisions will fundamentally shape your entire workflow. Careful planning of your objectives ensures that the chosen approach closely aligns with your scientific goals and will guide critical decisions about data annotation, model selection and performance evaluation.

Since segmentation serves as the central example in this chapter, it is important to first understand the different types of segmentation tasks encountered in microscopy before discussing how deep learning methods can be applied. These tasks can typically be categorised into three main types (Figure 9.1). Each segmentation type presents distinct challenges and is suited to different biological questions:

Binary segmentation: This is the [simplest form of segmentation](#) that separates the foreground from the background. For example, in a microscopy image, this involves distinguishing cell nuclei (foreground) from the rest of the image (background). This method is useful for detecting whether a structure is present or absent without distinguishing individual objects.

Instance segmentation: This [type of segmentation](#) identifies and labels each object independently. For instance, each cell in an image obtains a unique label. This method is crucial for tracking individual cells over time or measuring specific characteristics of each cell separately.

Semantic segmentation: This [segmentation strategy](#) involves labelling every pixel in an image according to its class, such as “nucleus,” “cytoplasm,” or “background.” Unlike instance segmentation, semantic segmentation does not differentiate between individual objects within the same class. This method is beneficial for analysing the spatial relationships and distribution of various cellular components.

Consider whether your segmentation solution is meant for a specific experiment or needs to generalise across various imaging techniques, sample types, or experimental conditions. Additionally, evaluate the volume of data to analyse, the feasibility of manual analysis, and the resources available to create a tailored image analysis pipeline. Avoid overengineering a solution when a simple analysis could provide the answer you seek.

Alongside task-specific considerations, it is equally important to clearly define the success criteria based on your objectives. For example, be prepared to answer the question, “*What do I need to accomplish for my analysis to be sufficient?*” – see Chapter 10 for more information. This is important because no analysis is ever 100% accurate. Establishing these criteria early streamlines both the development and evaluation processes, ensuring that your outcomes are scientifically meaningful and practically useful (see Chapter 10).

While the following steps focus on segmentation, the underlying principles can be readily adapted to a wide range of DL tasks in microscopy.

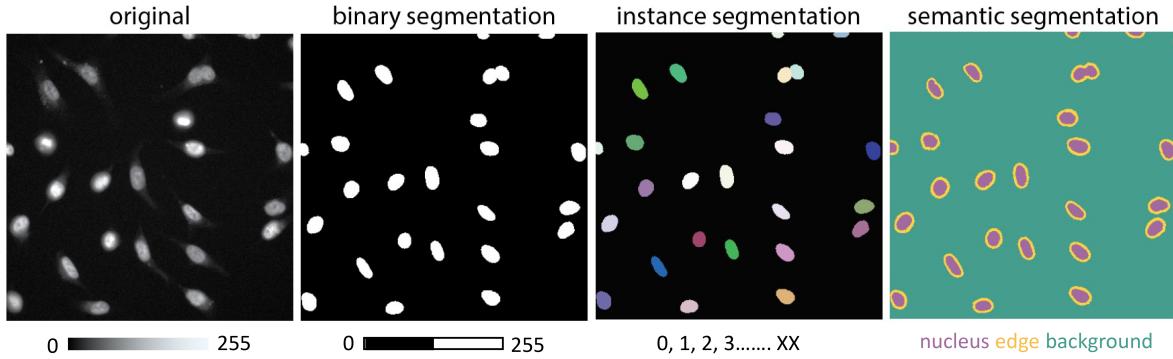


Figure 9.1: The three main types of segmentation in microscopy images. **original**: A raw grayscale fluorescence microscopy image showing cell nuclei stained with a nuclear marker. **binary segmentation**: Simplifies the image into two classes—foreground (white, nuclei) and background (black). **instance segmentation**: Assigns a unique label (shown in different colours) to each nucleus, facilitating individual object identification. **semantic segmentation**: Categorises each pixel into predefined classes—nucleus (purple), nucleus edge (yellow), and background (teal)—without distinguishing between individual objects.

9.2.2 Evaluating alternatives: Is DL the right choice?

Choosing the right computational method is essential for consistent and reproducible image analysis. For example in segmentation tasks, while DL can deliver exceptional segmentation performance, traditional methods and pixel classifiers still offer straightforward and efficient solutions for most tasks (Figure 9.2).

Traditional image processing techniques—such as intensity-based thresholding, morphological operations, edge detection, and filtering—are ideal for objects with clear, distinguishable features. These methods are well-documented, easy to understand, and usually require minimal computing resources. [Pixel classifiers](#), in particular, are user-friendly and can efficiently tackle many segmentation challenges with minimal manual annotation, making them highly effective for simpler analyses or smaller datasets.

DL methods excel in complex scenarios where traditional approaches fail, especially when dealing with noisy or context-dependent data. When trained on large, annotated datasets, DL models can effectively generalise across diverse imaging conditions and sample types, rapidly processing significant volumes of images. However, in the absence of pre-trained models, DL methods rarely offer shortcuts for data analysis. DL methods generally take effort and time to implement.

If you are unsure which approach to use, we usually recommend first trying classical image processing methods and pixel classifiers (Figure 9.1). We typically initiate a DL project only if these methods fail to produce satisfactory results (see Section 9.3.3.2).

Is DL the right choice for your segmentation project?

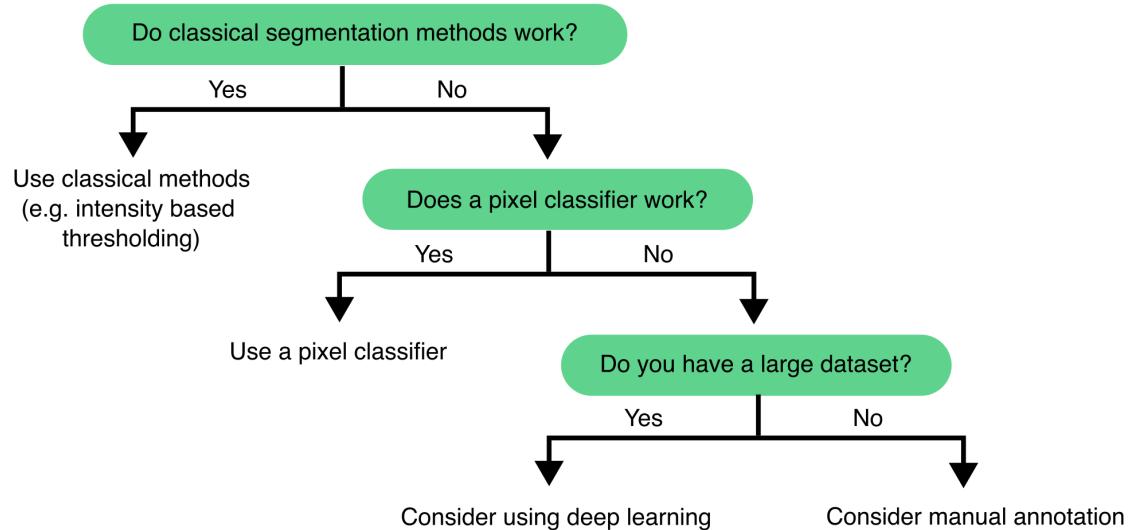


Figure 9.2: Is DL the right choice for your segmentation project? This decision tree guides the selection of appropriate segmentation approaches based on data complexity and project needs. Begin by testing classical image processing methods, such as intensity-based thresholding, which are efficient and easy to apply for well-defined features. If these methods prove insufficient, consider using a pixel classifier, which provides a user-friendly and effective solution for smaller datasets. Only consider DL if you possess a large annotated dataset and previous methods have failed. In the absence of suitable data or methods, manual annotation may be necessary.

9.3 Implementing a DL segmentation workflow

Although we use segmentation as our primary example, the workflow outlined in this section can be adapted to other deep learning tasks in microscopy and bioimage analysis.

9.3.1 Overview of a typical DL segmentation workflow

Once you decide to implement a DL approach for segmentation, the workflow can be divided into a series of steps (Figure 9.3).

The process starts by clearly defining your task (in this example, segmentation) and selecting the right DL approach (Step 1). Next, evaluate whether any existing pre-trained models can be used directly on your data or adapted for use (Step 2). If additional training is required—either from scratch or through transfer learning—prepare an appropriate training dataset that reflects your segmentation problem (Step 3). Then, train your model using the prepared dataset (Step 4) and thoroughly evaluate its performance using validation or test data (Step 5). Based on the results, you may need to refine the model by adjusting hyperparameters, improving annotations, or expanding the dataset. Once the model performs satisfactorily, it can be used to segment new, unseen data (Step 6). We will now discuss each step in more detail.

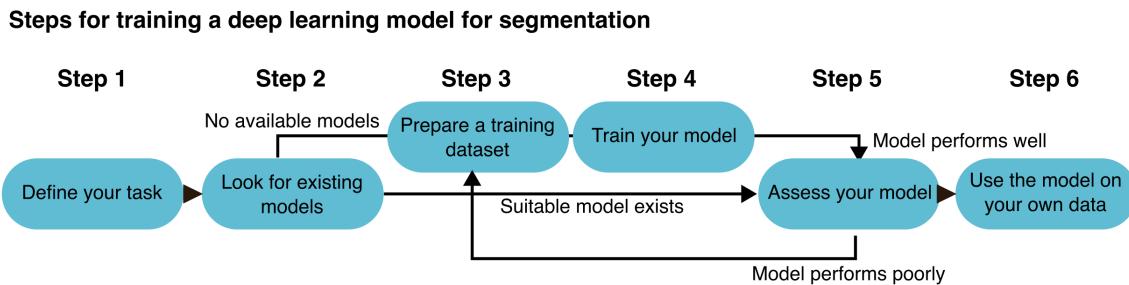


Figure 9.3: Conceptual workflow for training a DL segmentation model. The workflow begins with defining the segmentation task (Step 1), followed by searching for suitable pre-trained models (Step 2). If no such model exists, a training dataset must be prepared (Step 3), and the model is trained (Step 4). The trained model is then evaluated on validation or test data (Step 5). If it performs well, it can be applied to new data (Step 6); otherwise, the model is iteratively refined by returning to earlier steps. Feedback loops from evaluation to earlier stages help refine and improve the model accuracy.

9.3.2 Selecting a suitable DL approach

The first step in choosing a DL approach for image segmentation is to clearly define your segmentation task, whether it's binary, semantic, or instance segmentation (Figure 9.1), and to determine if you require 2D or 3D segmentation. Next, you should consider whether the model or tool you plan to use makes assumptions about the shapes or structures of the objects you want to segment. Understanding these assumptions will aid in selecting a model that fits your specific biological problem (see Section 9.2.1). Additionally, consider the amount of data that needs annotation for a particular DL approach (see Section 9.3.4.5). Finally, take into account your available computational resources (see Section 9.4.2). More complex models typically demand more GPU memory, longer training times, and additional storage, especially for 3D data or large datasets.

For example, StarDist [57], a widely used tool for nuclei segmentation, assumes that objects are **star-convex polygons**: i.e., a shape for which any two points on its boundary can be connected by a single line that does not intersect the boundary. This assumption works well for round or oval shapes but makes StarDist less suitable for segmenting irregularly shaped or elongated structures. In contrast, Cellpose [62] uses spatial vector flows to direct pixels toward object centres. This approach enables Cellpose to segment objects of various shapes and sizes, including irregular, elongated, or non-convex forms.

Choosing the right DL strategy requires aligning your goal, object shape, data dimensionality, and computing capacity with the strengths and assumptions of the available DL architectures.

9.3.3 Deciding whether to train a new model

9.3.3.1 Leveraging pre-trained models

The increasing availability of already trained (pre-trained) DL models has greatly simplified image analysis. Many of these models can be directly applied to your data, removing the need to train your own model [10, 20]. This reduces the technical barrier and saves time, making advanced analysis more accessible. However, it is essential to evaluate the quality of any pre-trained model before relying on its results (Figure 9.4). A model that performs well in one context may not be as effective on your specific data. Always conduct quality control by visually inspecting the outputs and assessing performance with quantitative metrics such as Intersection over Union (IoU) or F1-score, using a small, representative test set. This step is vital when model predictions are used in downstream analyses (see Section 9.3.7).

Another significant benefit of pre-trained models is their adaptability. Instead of starting from scratch, you can often fine-tune an existing model (see Section 9.3.6). This method entails retraining the model with a smaller, task-specific dataset, enabling it to adjust to your images while requiring far fewer annotations.

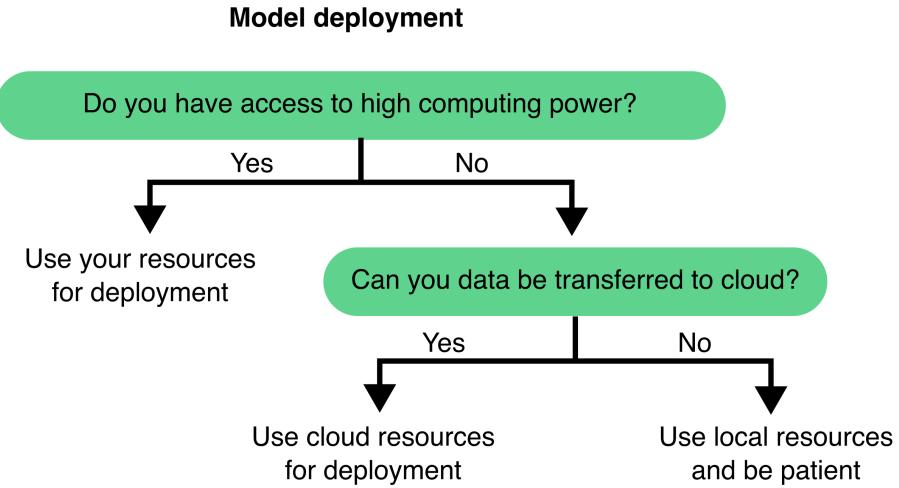


Figure 9.4: Decision workflow for selecting a DL model training approach. This flowchart outlines how to determine an appropriate training approach based on the availability and performance of pre-trained models.

Several excellent resources host pre-trained models suitable for microscopy. Researchers also increasingly share trained models alongside their datasets and publications, promoting open science. Platforms like Zenodo are commonly used for this purpose [19, 21], although deployment may require handling specific file formats or environments (see Chapter 8 for more information).

9.3.3.2 When to train your model

Pre-trained models serve as an excellent starting point for various microscopy tasks. However, there are many scenarios where training a custom model becomes essential. Custom training enables the model to learn the specific characteristics of your dataset, experiment, or imaging modality, resulting in enhanced performance [3, 46, 14]. This is particularly crucial when your data differs significantly from the data used to train existing models. Thus, their performance should always be validated. If quality assessment metrics are poor or key features are not accurately segmented, consider training your own model.

Ultimately, always evaluate the model's performance against your defined success criteria (see Section 9.3.7). Custom training may be the best path forward if the current model does not meet your needs.

9.3.4 Preparing your dataset for training

A well-designed training dataset is essential for developing successful DL models on tasks such as segmentation. The number of images and the quality of annotations needed vary based on factors such as task complexity and the architecture of the intended model.

9.3.4.1 Types of model training

For segmentation, most DL models are trained using supervised learning, where each input image is paired with a manually annotated ground truth mask. In this context, all objects that need segmentation must be annotated in the training dataset. This approach enables the model to learn a direct mapping from raw images to segmentation outputs (Figure 9.7).

However, alternative approaches can help reduce the need for extensive manual annotations:

- **Unsupervised learning** trains models without paired input and output data. Instead, the network identifies patterns or similarities in unlabelled images [31].
- **Self-supervised learning** involves designing tasks in which the model learns useful features directly from the input data without needing explicit labels [38].
- **Weakly supervised learning** uses partial, noisy, or imprecise labels to guide training, which can significantly reduce annotation effort [13, 42].

9.3.4.2 Creating Manual Annotations

Creating accurate annotations manually is time-consuming, particularly for 3D datasets. Tools like Fiji [54], Napari [1], and QuPath [9] are frequently employed for manual labelling. Typically, manual annotation involves drawing each object on the image and converting it into a mask or label.

💡 Here it is an example pipeline for manually annotating data using Fiji
[@fazeli2020]

1. **Open Fiji – activate the LOCI update site and restart Fiji.**
LOCI tools are required for exporting ROI maps. To enable them, go to *Help > Update > Manage Update Sites*, look for ‘LOCI’ and check the *Active* checkbox. Then, click on *Apply and Close* and *Apply Changes*, this update site ensures the necessary plugins are installed. Finally, restart Fiji.
2. **Open your image you wish to annotate.**
Use *File > Open* to browse and load the microscopy image that you want to label

manually. You can also drag and drop your image to Fiji.

3. Select the Oval or Freehand selection tool.

These tools, found in the Fiji toolbar, allow you to manually outline the structures of interest in your image.

4. Start drawing around each object (yes, each one!).

Carefully trace each cell or feature you want to annotate—precision is key to ensure useful training data for DL.

5. After drawing each object, press “t” on your keyboard → the selection will be stored in the ROI manager.

This adds the drawn region to the ROI (Region of Interest) list, keeping track of all annotated objects in the image.

6. Repeat until all objects are in the ROI manager.

Continue drawing and pressing “t” until you have annotated every relevant object in the image.

7. When finished, go to *Plugins* › *LOCI* › *ROI Map*.

This plugin converts all saved ROIs into a single labeled ROI map image, assigning unique values to each region.

8. Save the generated ROI map with the same title as the original image in a separate folder.

Consistent naming ensures each annotated map can be correctly matched with its corresponding raw image during training or analysis.

9. At the end, you will have one folder with the original images and another for the ROI maps.

This separation makes it easier to organise and use your data with image analysis or DL pipelines.

9.3.4.3 Accelerating annotation with automatic initial segmentations

Creating high-quality annotations often represents the most time-consuming aspect of training a DL model for segmentation. To alleviate this burden, you can start from automatically produced initial segmentations. For example, using simple thresholding methods such as Otsu’s thresholding to generate rough segmentations can decrease the total annotation time. Even more powerfully, pre-trained DL models such as those provided with StarDist [57] and Cellpose [62] can generate more accurate initial segmentations that users can manually refine. These annotations can then be used to retrain the model, establishing an iterative cycle that accelerates both labelling and model refinement.

New tools are also pushing the boundaries of interactive annotation. For example, Segment Anything for Microscopy (SAM) [3] facilitates automatic and user-guided segmentation and allows the model to be retrained on user-provided data. Similarly, Cellpose 2.0 [46] features a human-in-the-loop workflow, allowing users to edit DL-generated segmentations. This hybrid approach enhances accuracy while significantly reducing the time and effort required for manual annotation.

9.3.4.4 Expanding your dataset with augmentation and synthetic data

When the number of training samples is limited, [augmentation](#) techniques can enhance dataset diversity to improve the model’s generalisation ability and performance on validation and testing [35, 59]. Common augmentation strategies include image rotation, flipping, scaling, and contrast adjustment. However, it’s important to apply augmentation carefully, as excessive or unrealistic augmentation can confuse the model or cause it to learn patterns that do not exist in real data.

In the absence of sufficient real data, synthetic data generated through simulations or [domain randomization](#) can help pre-training a model [37, 49]. These synthetic samples can expose the model to a broader range of scenarios early in training, before transitioning to fine-tuning with real, annotated data.

In summary, a successful segmentation pipeline relies on a careful balance between data quantity and annotation quality. Augmentation strategies can efficiently help to scale and balance training datasets.

9.3.4.5 Choosing the dataset size: specific vs. general models

In supervised training, it is crucial that each image in the training set is accompanied by a corresponding label image (see Section 9.3.4.1). The number of image-label pairs required depends on the number of labels per image, the complexity of the model and the desired level of generalisability. Still, the key is having enough representative examples and corresponding annotations for the model to learn meaningful patterns.

Small and well-curated datasets consisting of tens of images may suffice for highly specific applications, such as segmenting cells or nuclei using a defined imaging modality [14]. In these scenarios, [transfer learning](#) can also be especially beneficial (see Section 9.3.6). Models designed to generalise across a wide range of conditions, tissue types, or imaging modalities typically require much larger and more diverse datasets (hundreds to thousands of annotated images) [62]. These datasets are essential for capturing the inherent variability in broader use cases.

9.3.5 Training a segmentation model from scratch

Once you have annotated your training dataset, the next steps are to organise your data for training, initialise your model by selecting appropriate [hyperparameters](#), and start the training process (Figure 9.7).

9.3.5.1 Splitting your training data: training, validation, and test sets

A crucial part of preparing your dataset is dividing it into three subsets: training, validation, and test sets. Each subset should contain the original microscopy images paired with their corresponding ground truth segmentations. A common strategy is to allocate 70–80% of the data for training, 10–15% for validation, and the remainder for testing. To ensure unbiased evaluation, ensure these subsets do not overlap in terms of fields of view, represent the variability of your entire dataset, and are randomly assigned to each set respectively.

The **training set** is used to train the model to recognise relevant features. To enhance generalisation, it must encompass a broad spectrum of scenarios and image conditions. Otherwise, the model risks overfitting—excelling with the training data but faltering with new images (Figure 9.6).

The **validation set** is used during training to provide feedback on the model’s performance with unseen data. This feedback, conveyed as validation loss, assists in detecting overfitting (Figure 9.6), guiding hyperparameter tuning (see Section 9.3.5.3), and informing training decisions. Although a separate validation set is ideal, many workflows create one in practice by reserving a portion (typically 10% to 30%) of the training data.

The **test set**, which serves a separate role, evaluates the model’s performance on entirely unseen data. Unlike the validation set, the test set is not utilised during training, ensuring an unbiased performance assessment. Test images should also include ground truth annotations to facilitate quantitative quality control. Reporting test set performance, using metrics such as accuracy, IoU, or F1-score, is crucial, especially when publishing or benchmarking your model [34].

9.3.5.2 Understanding the training process

A DL model is composed of multiple layers (Figure 9.5). Each layer contains tens to hundreds of image processing operations (typically multiplications or convolutions), each controlled by multiple adjustable parameters (called weights). Altogether, a DL model may contain millions of adjustable weights. When an input image is processed by a DL model, it is sequentially processed by each layer until an output is generated. Segmentation tasks typically involve converting input images into labelled outputs. During training, the model weights are modified as the model learns how to perform a specific task.

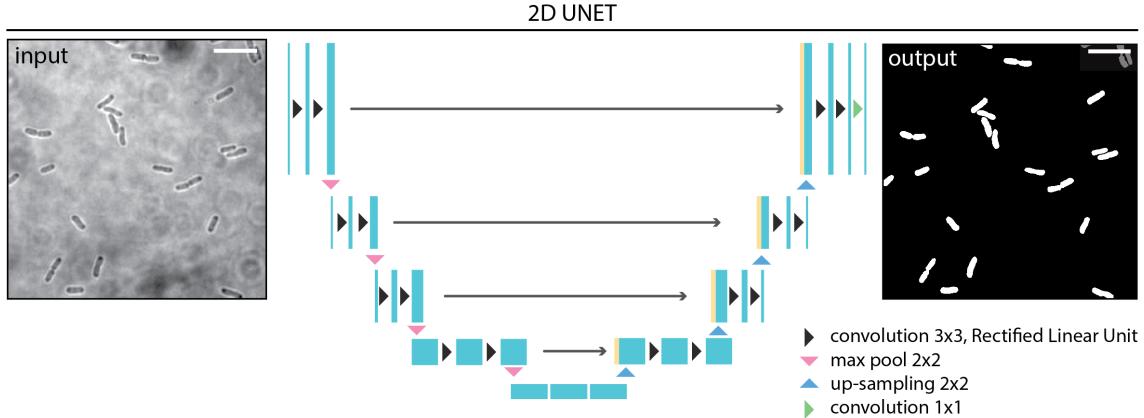


Figure 9.5: 2D U-Net architecture for image segmentation. It applies layers of convolutions, pooling, and upsampling to extract features and generate labelled segmentation masks. During training, model weights are iteratively adjusted based on the difference between predictions and ground truth labels, using a loss function and backpropagation.

Training begins with initialising these weights. When training from scratch, the initialisation is often random. However, when using a pre-trained model, the weights are already optimized based on previous training, providing the model with a significant head start (see Section 9.3.6).

The training process is iterative (Figure 9.7). Each cycle of training is called an [epoch](#). During each epoch, the model typically learns from every image in the training set. Since datasets are often too large to fit into memory all at once, each epoch is divided into steps or iterations, with each step processing a smaller subset of the data known as a batch. The batch size determines how many samples are processed simultaneously.

During each step, the model generates predictions for the current data batch. These predictions are compared to the ground truth labels using a loss function that calculates the similarity between the predictions and the ground truths. This score is called the training loss. The model utilises this feedback to adjust its weights through a process known as [backpropagation](#), guided by an optimisation algorithm, to improve its accuracy in future iterations.

At the end of each epoch, the model assesses its performance on the validation set, which comprises data it has not encountered during training. This produces the validation loss, indicating how well the model generalises to new data.

Monitoring both training and validation losses during training helps determine whether the model is learning effectively. A consistently decreasing validation loss indicates that the model is improving and generalising well (see Section 9.3.5.4).

9.3.5.3 Choosing your model hyperparameters

Now that you understand the training process, the next step is to configure the model’s hyperparameters, which are the settings that dictate how the model learns. While the model’s parameters (its weights) are updated during training, hyperparameters are established beforehand, defining the structure and behaviour of the training process. Below are some of the most common hyperparameters and their effects on training:

- **Batch size:** This refers to the number of images processed simultaneously in each training step. Smaller batch sizes are less demanding on memory and may enhance generalisation, although they can result in slower training. In contrast, larger batch sizes accelerate training but necessitate more GPU memory.
- **Epochs:** An epoch refers to a training cycle in which the model processes the entire training dataset. Increasing the number of epochs allows the model to learn more, but also raises the risk of overfitting. More is not always better; it is essential to monitor performance on the validation set.
- **Learning rate:** It determines the extent to which the model’s weights are adjusted during training. A high learning rate can result in quicker training but may overshoot the optimal solution. Conversely, a low learning rate provides more stable progress, although it may slow down convergence.
- **Optimizer:** An algorithm that updates weights to minimise the loss function. Common optimisers include SGD (stochastic gradient descent) and Adam (adaptive moment estimation), the latter being widely used for its adaptive learning rate and robust performance.
- **Learning rate scheduler:** Dynamically adjusts the learning rate during training, typically decreasing it after a specific number of epochs or when the validation loss plateaus. This approach helps balance rapid early learning with more refined convergence later on.
- **Patch size:** Instead of using full-resolution images, smaller patches are often utilised for training to reduce memory usage and enhance training speed. The patch size is determined by both available resources and the scale of the structures to be segmented.
- **Patience (early stopping):** This parameter defines the number of epochs to wait before halting training if the validation loss does not improve. It helps prevent wasting resources on overfitting and overtraining.

Given the many possible configurations, tuning hyperparameters is often essential—especially when applying a model to new data. Start with the recommended values from the model’s original publication, but you might need to conduct a hyperparameter search to optimize performance. This can range from a simple grid search to more advanced methods, such as Gaussian process-based Bayesian optimisation [28] or genetic algorithms [2].

9.3.5.4 Monitoring training and validation Losses

Once your model begins training, it is helpful to evaluate its learning progress. The two key metrics for assessment are the training loss and the validation loss (Figure 9.6). Monitoring both throughout the training process offers insight into whether your model is improving and learning to generalise beyond the training data. The three main behaviours that you may encounter during training are:



Figure 9.6: Monitoring training and validation losses during model training. The plot illustrates three typical learning behaviours: **underfitting** (both losses remain high and similar), **good fitting** (both losses decrease, with validation loss slightly higher), and **overfitting** (training loss continues to decrease while validation loss plateaus or increases), highlighting the importance of tracking these metrics to assess model performance and generalisation.

- **Underfitting:** The model has been trained with insufficient data or for too few epochs, resulting in similar training and validation losses, which is far from optimal.
- **Good fitting:** Both training and validation losses decrease, with the validation loss slightly higher (worse) than the training loss, which is expected. This represents the ideal scenario.
- **Overfitting:** The model achieves an excellent training loss, but the validation loss does not improve and may in fact diverge. This may indicate overly similar training data or excessive training epochs, preventing the model from generalising to new data.

9.3.6 Fine-Tuning Pre-existing Models

Instead of training a model from scratch, fine-tuning an existing DL model is usually more efficient, especially when your data resembles the dataset used to train the original model. This

approach utilises pre-trained weights and previously learned features, significantly decreasing the amount of required annotated data, training time, and computational resources.

9.3.6.1 Applying Transfer Learning

Transfer learning refers to the process of taking a pre-trained model and adapting it to a new but related task by providing task-specific training data, typically in the form of manually annotated image pairs (Figure 9.7). Transfer learning typically involves freezing part of the model (for instance, the initial layers or all layers except the last ones), so their weights are not updated during training. Only the unfrozen layers are updated when the model is trained on new data. Then, the model is trained on the new data, but only the layers that you have unfrozen will be updated. Since the base model already encodes many useful low-level features (e.g., edges, shapes, textures), this approach allows researchers to focus on refining the model for their specific biological structures or imaging modalities [36, 41].

This method is especially effective when:

- You have limited training data available.
- Your imaging conditions closely match those of the pre-trained model.
- You wish to quickly adapt a general model to a specific dataset.

9.3.6.2 Conducting fine-tuning

In classic fine-tuning, all layers of the pre-trained model are retrained, with their weights initialised from the original training (Figure 9.7) [3, 46]. Thus, you continue training the full model using the new data. This approach allows the model to adjust more comprehensively to new data while still preserving the advantages of pre-learned features.

Classic fine-tuning is ideal when:

- Your dataset is moderately different from the original training data (e.g., the same biological structure but different staining or modality).
- You expect that earlier layers may need to adapt, not just the final classifier or output layers. Early layers in deep networks typically learn to detect general features such as edges, textures, or simple shapes, while later layers capture more complex, task-specific patterns. If your new data differs in basic appearance or imaging modality, updating the early layers helps the model better extract relevant low-level features from your images.

- You have enough annotated data to avoid overfitting during full model training. Although this method is more computationally demanding than transfer learning, where only a subset of layers are retrained, it often leads to better results on diverse datasets..

9.3.6.3 Iterative training: keeping humans in the loop

Iterative fine-tuning is an interactive approach that combines model prediction with human annotation (see Section 9.3.4.3). The workflow typically starts with a pre-trained model predicting new images. A user then manually corrects or annotates these predictions, and the improved annotations update the model (Figure 9.7). This cycle continues, progressively enhancing the model’s accuracy with each iteration until it performs as expected [16, 18, 46].

This method is particularly powerful when:

- Annotated data is scarce or expensive to generate.
- You work with rare structures, unusual imaging conditions, or new experimental systems.
- You want to efficiently build a custom model using feedback from domain experts.

9.3.7 Evaluating the performance of your model

With the rapid increase in DL tools and pre-trained models, it has become easier to use DL for image segmentation, but harder to determine which model will work best for your data. Regardless of how promising a model appears, you must always evaluate its performance before trusting its results. Evaluation is not optional; it is a critical step to ensure that the model meets the requirements of your specific task [34] (Figure 9.8).

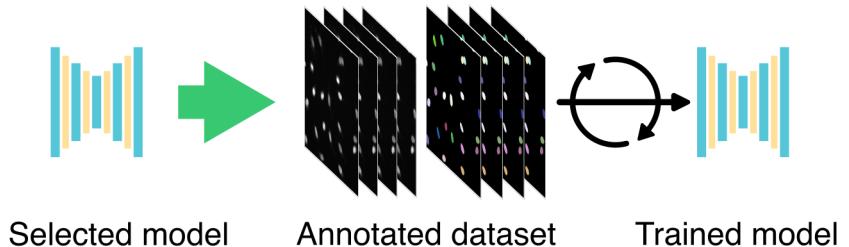
There are two main ways to evaluate a model:

- Qualitative evaluation entails visually inspecting the model’s predictions. This approach can help you quickly identify clear errors or failures. It is effective for a small number of images, but it becomes impractical for large datasets or for comparing similar-looking outputs across multiple models.
- Quantitative evaluation provides objective metrics for comparing models and tracking improvements. To achieve this, you need a small, labelled test set (typically 5 to 10 images with accurate ground truth segmentations). This test set must remain independent of your training and validation data to ensure an unbiased assessment.

Common metrics used in quantitative evaluation include:

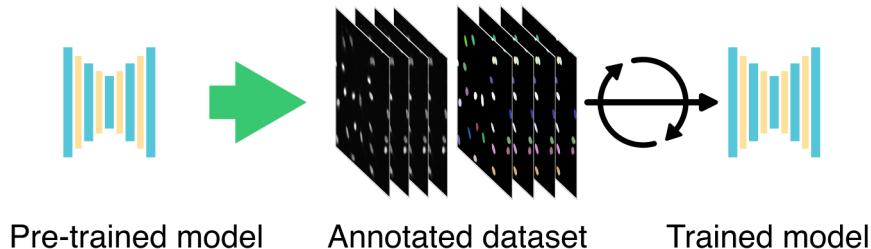
Training from Scratch

A model is trained from randomly initialized weights using a large annotation dataset



Transfer Learning or Fine-Tuning

A pre-trained model is adapted to a new task by adding new annotated data



Human-in-the-Loop Fine-Tuning

Model predictions are iteratively corrected through manual annotations

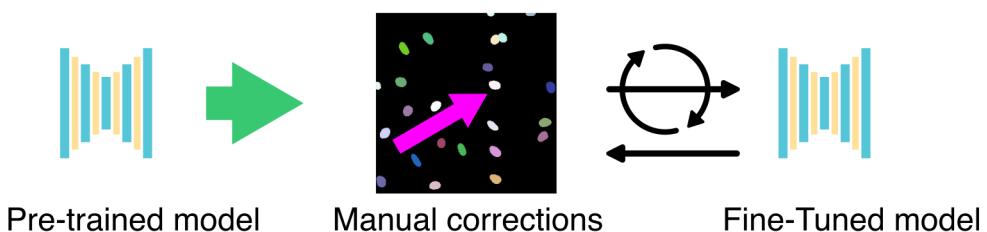


Figure 9.7: Strategies for training DL models for image segmentation. A model can be **trained from scratch** using a large annotated dataset, **fine-tuned** from a pre-trained model with task-specific data, or refined through **human-in-the-loop** workflows where model predictions are manually corrected and fed back for retraining. These approaches balance performance, data availability, and annotation effort.

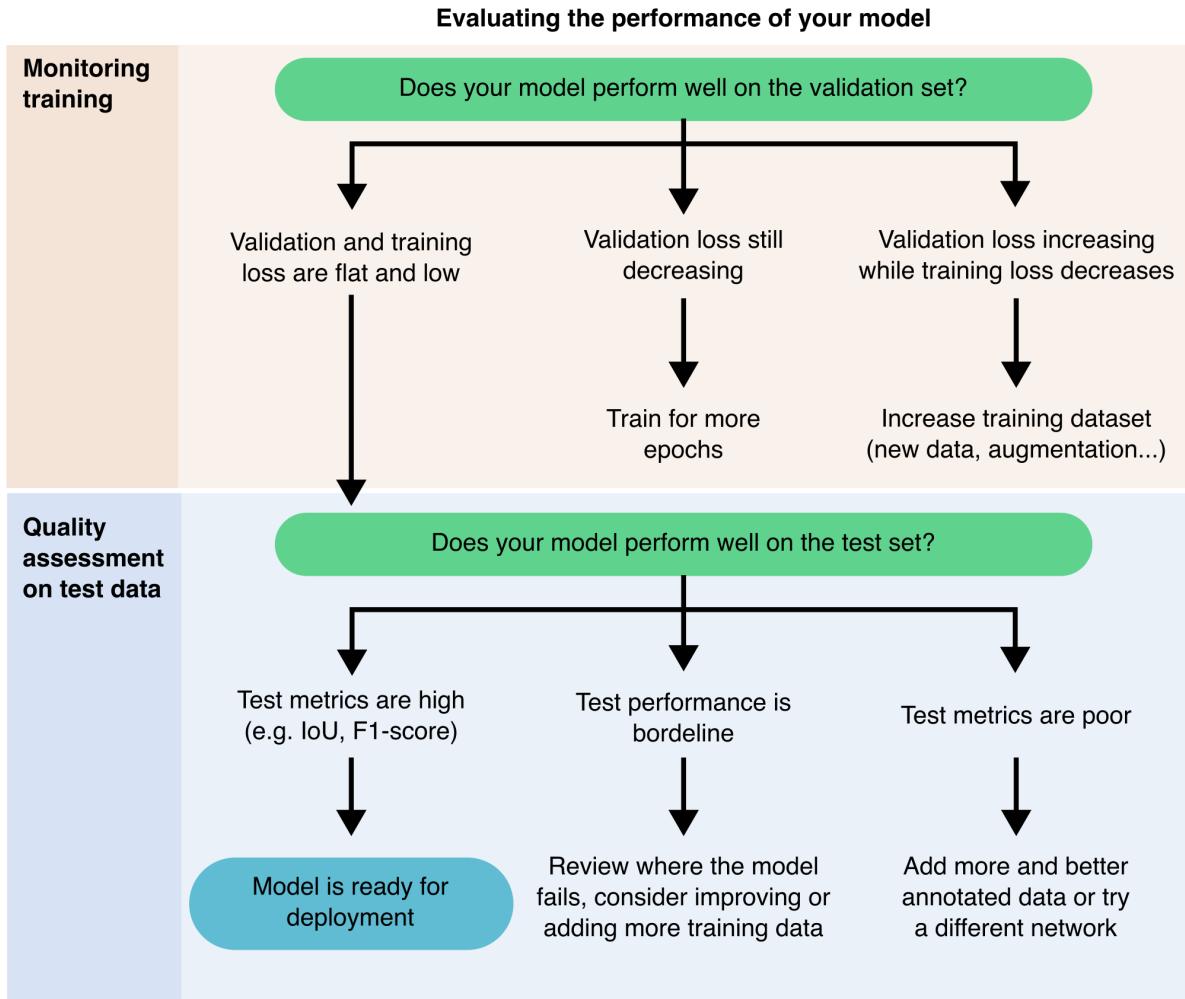


Figure 9.8: Workflow for evaluating DL model performance during training and on test data. Evaluating model performance is essential before deploying any DL model for image segmentation. This diagram outlines a two-stage process: assessment during training and on a separate test set. During training, validation and training losses (see Section 9.3.5.4) guide whether to continue training, stop, or expand the dataset. After training, performance is evaluated using a test set. High test metrics (e.g., IoU, F1-score) indicate readiness for deployment. Borderline or poor results suggest reviewing errors, refining training data, or trying a different model. This approach ensures model reliability and task-specific performance.

- Intersection over Union (IoU), also known as the Jaccard Index, measures the overlap between the predicted segmentation and the ground truth.
- F1-score (Dice coefficient): This is especially valuable when the object of interest covers a small area in the image, as it balances precision and recall.
- **True Positives** (TP), **False Positives** (FP), and **False Negatives** (FN) are particularly important in semantic segmentation and can be used to calculate the **IoU** or **F1 score**.

For more information on these metrics, we recommend [34] (also see Chapter 10 for more information).

If a model fails to produce reasonable results, even on simple examples, you can often reject it based solely on qualitative inspection. However, in these cases, quantitative metrics can still help you understand how and where the model fails.

If your evaluation metrics indicate weak performance, especially for certain structures or image types, you may need to fine-tune the model (see Section 9.3.6). Consistently strong scores across various test images suggest that a model could be dependable and ready for deployment. If no pre-trained model meets your expectations, the best course may be to train your model using your images (see Section 9.3.5).

In summary, **never skip evaluation**. Every model must be tested—both visually and quantitatively—to ensure it truly works for your data and provides results you can trust.

9.3.8 Deploying your model on new data

Once a segmentation model has been trained and validated, it can be used on new, unseen images. This step typically involves feeding new images into the model to generate segmentation predictions. The deployment approach relies on the computational resources (see Section 9.4.2) as well as the size and complexity of your dataset (Figure 9.9).

9.3.9 Troubleshooting Common Problems

I found a tool or DL model online, but it does not work. What should I do?

When should I train a model or segment manually?

Refer to (Section 9.3.3.2) for more details, but generally, this decision depends on your dataset and the performance of existing pre-trained models (Figure 9.10). If you only need to segment a small number of images, manually segmenting them is often the quickest and simplest solution. However, if you are dealing with a large dataset, it may be more efficient to annotate a small subset and use it to train a deep-learning model that can automate the segmentation of the rest.

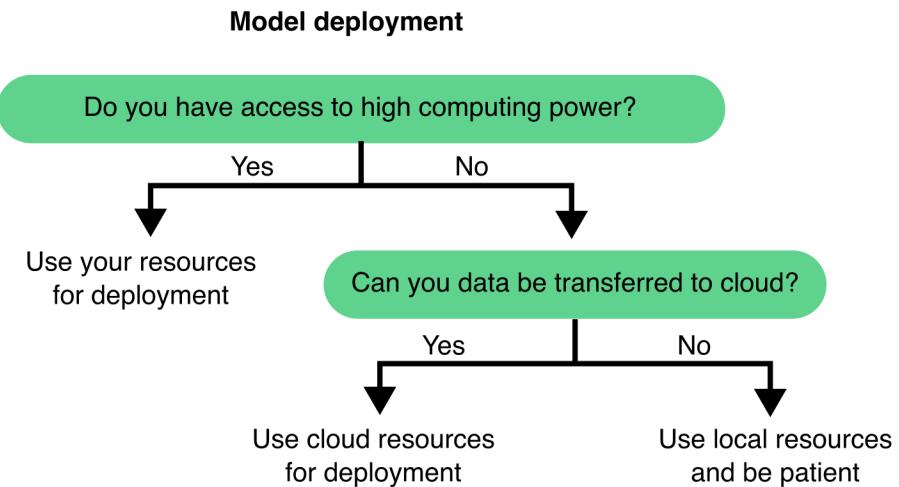


Figure 9.9: Decision workflow for model deployment strategy based on computational resources. The choice of deployment strategy depends on the availability of computational resources (see Section 9.4.2) and the sensitivity of the data . If high-performance computing resources are available locally, these should be used for deployment. In their absence, consider whether the data can be transferred to the cloud. If so, cloud-based resources offer an efficient solution. However, if data transfer is restricted—due to size or sensitivity—local deployment remains the only option, though it may require significantly more time.

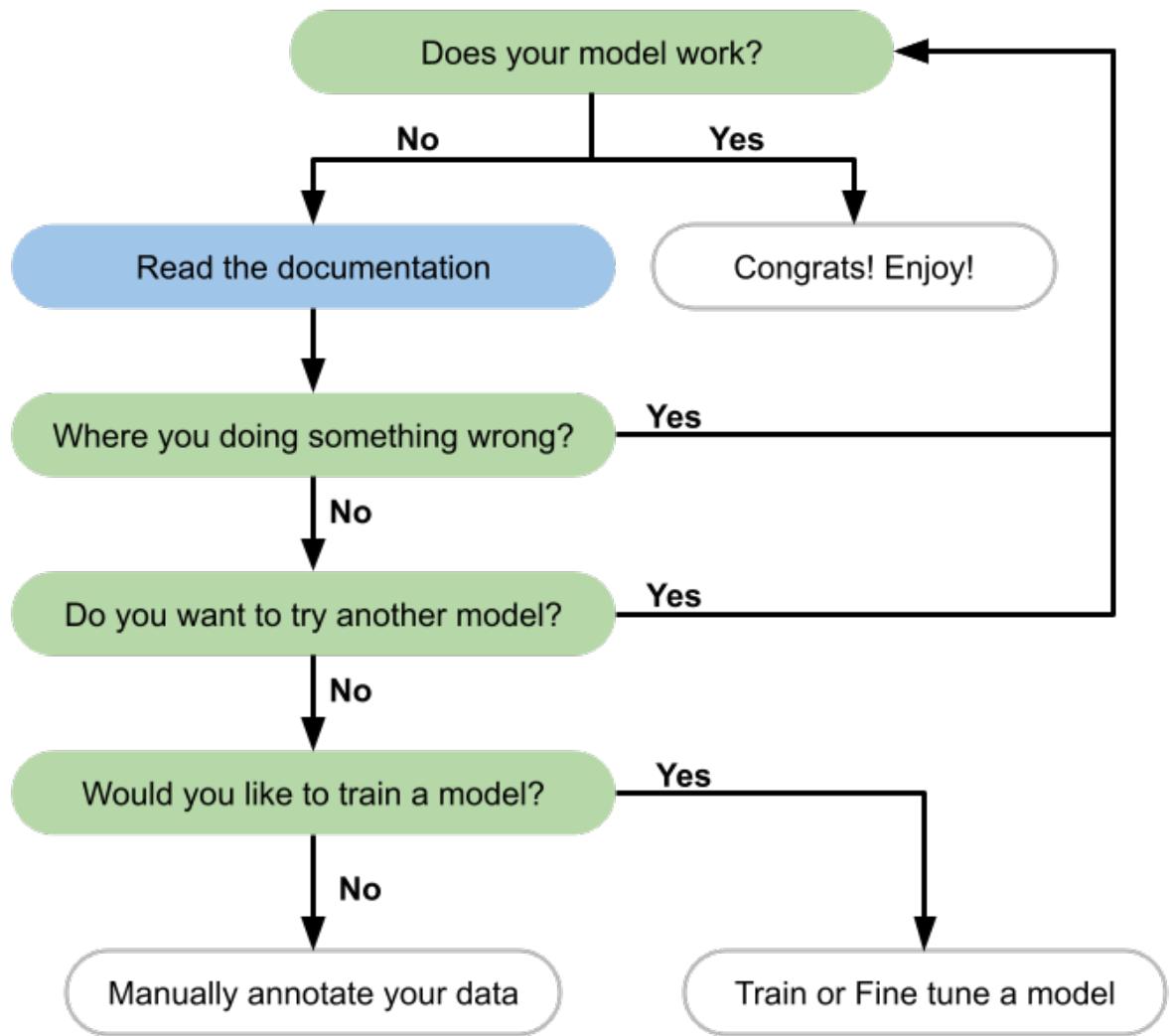


Figure 9.10: Common DL segmentation problems and troubleshooting tips.

I decided to train my DL model, but it is not performing correctly. What should I do?

First, ensure that you have trained the model for a sufficient number of epochs—this depends on the size of your dataset and the architecture of the model. Check the training loss: if it has plateaued, your model may be fully trained. If it is still decreasing, continue training.

If training is completed but results are poor, examine your data. Is the model missing specific features? Are there types of cells or structures that it consistently fails to segment? If so, ensure those examples are well represented and correctly annotated in your training data. You may need to enhance or expand your annotations.

If performance is poor, you may need additional annotated data to help the model generalise more effectively (Figure 9.8). Consider the following questions:

- Is my dataset balanced? Does it include sufficient examples of each structure or class I want to segment?
- Am I training on one experimental batch while validating or testing on another?

How many images should I have to train my model?

Refer to Section 9.2.1 for more details. There's no one-size-fits-all answer—it depends on the complexity of your task, your model architecture, and the variability in your data. More complex tasks typically require more data. Larger images can also be broken into more patches, effectively increasing your dataset size. While few-shot models are being developed for small datasets, most established DL models require a substantial amount of data.

Possible technical issues that you may encounter when training your DL model.

- The model predicts the same class for all pixels or segments in every cell. Your dataset might be unbalanced, containing too many similar examples. Adding more diverse or underrepresented examples can help the model learn to differentiate between classes.
- Out-of-memory errors during training: Consider reducing the batch size or the image patch size. If that doesn't resolve the issue, consider switching to a workstation or cloud service with greater computational capacity.
- The model performs well on training data but poorly on new images, suggesting overfitting (Figure 9.6). Implement data augmentation and increase dataset diversity to help the model generalise better.
- Inconsistent results across different computers: Differences in GPUs or environments can cause slight variations in outcomes. If the differences are significant, verify that all systems use consistent software versions and configurations. For further information on this topic, refer to Section 9.4.3.

9.4 Further considerations for DL segmentation

9.4.1 Choosing the Right Tools for DL

Selecting the right tools to train and use DL models depends mainly on your level of programming experience and comfort with technical interfaces.

If you prefer not to write code or use command-line tools, opt for platforms that offer graphical user interfaces (GUIs) or interactive notebooks with pre-configured workflows. These tools let you perform powerful segmentation tasks using intuitive interfaces and simple widgets.

GUI-based tools include, for instance (see Chapter 8 for more tools):

- Cellpose GUI
- Fiji with DeepImageJ and StarDist plugins
- Napari
- Ilastik
- QuPath

Interactive Jupyter notebooks provide a flexible balance between code and GUI. They enable you to execute code in manageable steps (cells) and immediately see the results. Tools like BiaPy, and DL4MicEverywhere [26] leverage Jupyter notebooks, concealing complex code behind user-friendly interfaces. These platforms cater to users with little or no coding experience while still allowing advanced users to access and modify code as needed. DL4MicEverywhere, in particular, established a widely adopted framework for training DL models via notebooks, contributing to the standardisation and simplification of the workflow.

If you are comfortable with programming, you will have even more flexibility. Languages such as Python, MATLAB, Julia, Java, and Rust provide options for building and customizing DL workflows. Python stands out as the most beginner-friendly and widely supported choice, boasting a large ecosystem of libraries and community support. Popular Python libraries for DL include PyTorch, TensorFlow, Keras, and JAX.

While coding can involve a steeper learning curve, it allows you to create customized pipelines, integrate various tools, and troubleshoot intricate workflows, unlocking the full potential of DL for microscopy segmentation.

9.4.2 Managing Computational Resources

When using DL for microscopy, an important consideration is the availability and capacity of your computational resources (Figure 9.9). High-performance DL models, particularly those used for 3D image data, can be very demanding regarding memory and processing power.

When selecting or designing a DL model, evaluate your available infrastructure:

- GPU memory: Determines how large your model and batch size can be.
- Training time: Influences your ability to iterate quickly; simpler models train faster.
- Dataset size: Larger datasets benefit from more powerful hardware and longer training times.

A practical strategy involves starting with lightweight models that demand fewer resources and scaling up to more complex architectures only if performance improvements become necessary. Tools like StarDist and Cellpose, for example, provide efficient options that function effectively with relatively modest hardware.

Additionally, consider whether to train and deploy your model locally or in the cloud (Figure 9.11). Local training is often more feasible if you already have access to a compatible workstation and want full control over data and execution. However, cloud-based services like Google Colab or AWS offer access to more powerful hardware, removing the need for local infrastructure—this is especially beneficial when working with large models or 3D datasets.

There are four typical combinations of training and prediction workflows:

- Training and prediction locally is well suited for small to medium-sized datasets, especially when computational demands are moderate and data privacy is a priority. This approach also supports some user-friendly desktop applications, such as the Cellpose 2.0 [46], which can be run locally without requiring cloud access or advanced technical setup.
- Training locally, prediction in the cloud may be useful when models are trained in-house but need to be deployed at scale or integrated into cloud-based pipelines.
- Training in the cloud, prediction locally enables researchers to take advantage of powerful cloud GPUs for model development, while keeping inference close to the data source (e.g., a microscope workstation or in case of sensitive data).
- Training and prediction in the cloud is well suited for collaborative projects or large-scale deployments, where access to centralized, scalable infrastructure is critical.

Choosing between these strategies depends on your data size, hardware access, choice of software, collaboration needs, and whether your workflow prioritizes flexibility, scalability, or control.

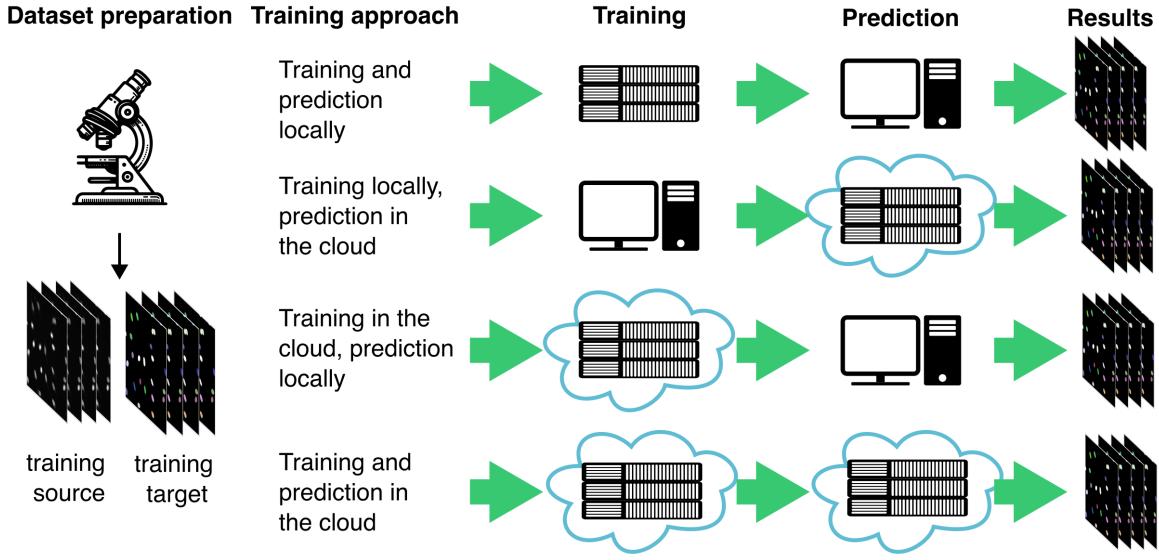


Figure 9.11: Training and deployment strategies for DL models in microscopy. Depending on the available tools and infrastructure, models can be trained and deployed locally or in the cloud. Modified from [14].

9.4.3 Ensuring Reproducibility in DL

When sharing how you trained a DL model, two key elements often come to mind: the dataset used and the code that runs the model. However, in practice, reproducibility extends beyond just data and code. In programming environments like Python, which rely heavily on external libraries, ensuring reproducibility also requires capturing the exact configuration of the environment in which the model was trained.

DL models are sensitive to changes in library versions and dependencies. Even minor differences in the software stack can result in inconsistent outcomes or training failures. While sharing a list of dependencies (e.g., a requirements.txt or a Conda environment file) is a constructive step, differences in operating systems or local setups can still lead to issues.

A robust and increasingly popular solution is containerisation. Containers package software, dependencies, and environment settings into a portable and self-contained unit. One of the most widely used containerization tools is Docker. A Docker container can be considered a lightweight, standalone [virtual machine](#) that includes everything needed to run code, such as the operating system, libraries, and runtime, ensuring applications run consistently across different machines.

Using containers ensures that your model training and inference processes remain consistent, no matter who executes them or where they are conducted. This greatly simplifies the ability

of collaborators or reviewers to reproduce your results.

For researchers unfamiliar with software development, tools like DL4MicEverywhere [26] and bia-binder [52] simplify the use of containers by integrating them into user-friendly Jupyter notebook environments. These platforms enable researchers to benefit from the reproducibility of containers without needing to manage complex setups or command-line tools.

Reproducibility is crucial for establishing trust in computational results and facilitating long-term scientific collaboration. To ensure your DL workflows are reproducible, follow these best practices:

- Pin every software version used in your workflow.
- Document your environment setup thoroughly.
- Provide a containerised version of your training and inference pipeline when possible.
- Taking these steps will make it easier for others to reproduce your results, build on your work, and apply your models in different research settings.

For more information on best practices, consult [34].

9.5 Summary & Outlook

Segmenting microscopy images remains a critical yet challenging task in bioimage analysis. In this chapter, we have used segmentation as a representative example to illustrate deep learning workflows and considerations. However, the strategies and best practices described here—such as data preparation, model selection, training, evaluation, and deployment—are relevant to a wide range of image analysis tasks, including classification, detection, and tracking. DL has undeniably transformed this field, offering robust solutions for segmenting complex and variable structures. However, as this chapter emphasizes, DL is not always the fastest or the best approach. Classical image processing techniques or pixel classifiers often provide faster, simpler, and highly effective alternatives in many scenarios.

The decision to use DL should be driven by the complexity of the task, the availability of annotated data, and the specific goals of the segmentation project. Successful DL implementations often require significant investments in data curation, annotation, and computational resources. Furthermore, training from scratch is frequently avoidable thanks to the growing ecosystem of pre-trained models and resources shared by the community.

Notably, the landscape of DL segmentation is rapidly evolving. The emergence of foundation models, which are large, versatile networks pre-trained on vast and diverse datasets, promises to further lower the barriers to entry [3]. These models enable transfer learning, fine-tuning, and even zero-shot segmentation, where accurate predictions can be made on previously unseen

data with minimal or no task-specific training. This shift opens exciting new avenues for researchers who previously lacked the resources or expertise to apply DL in their work.

The ongoing development and democratization of DL tools, along with enhancements in model generalisability, human-in-the-loop workflows, and reproducibility, are changing how microscopy data is analyzed. Still, the key to successful segmentation will always involve careful planning, quality control, and selecting the right tool for the task, whether it involves DL or not.

10 Output Quality

Through the Lens of Segmentation

Under your first header, include a brief introduction to your chapter.

Starting prompt for this chapter: Chapter 10 addresses how to assess the quality of a model's output, mentioning Metrics Reloaded. This chapter should address the question: how do I know my model is good enough? It should frame this discussion using the example of a segmentation model and discuss how tools can identify uncertain decisions from a model.

10.1 Include section headers as appropriate

Use markdown heading level two for section headers. You can use standard markdown formatting, for example *emphasize the end of this sentence*.

This is a new paragraph with more text. Your paragraphs can cross reference other items, such as Figure 11.1. Use `fig` to reference figures, and `eq` to reference equations, such as Equation 11.1.

10.1.1 Sub-subsection headers are also available

To make your sections cross reference-able throughout the book, include a section reference, as shown in the header for Section 11.4.

10.2 Bibliography and Citations

To cite a research article, add it to references.bib and then refer to the citation key. For example, reference Stringer et al. [62] refers to CellPose and reference Chamier et al. [14] refers to ZeroCostDL4Mic.

10.3 Adding to the Glossary

We are using R code to create a glossary for this book. To add a definition, edit the glossary.yml file. To reference the glossary, enclose the word as in these examples: LLMs suffer from [hallucinations](#). It is important to understand the underlying [training data](#), [validation data](#) and [false positives](#) to interpret your results. Clicking on the word will reveal its definition by taking you to the entry on the [Glossary](#) page. Pressing back in your browser will return you to your previous place in the textbook.

10.4 Code and Equations

This is an example of including a python snippet that generates a figure

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

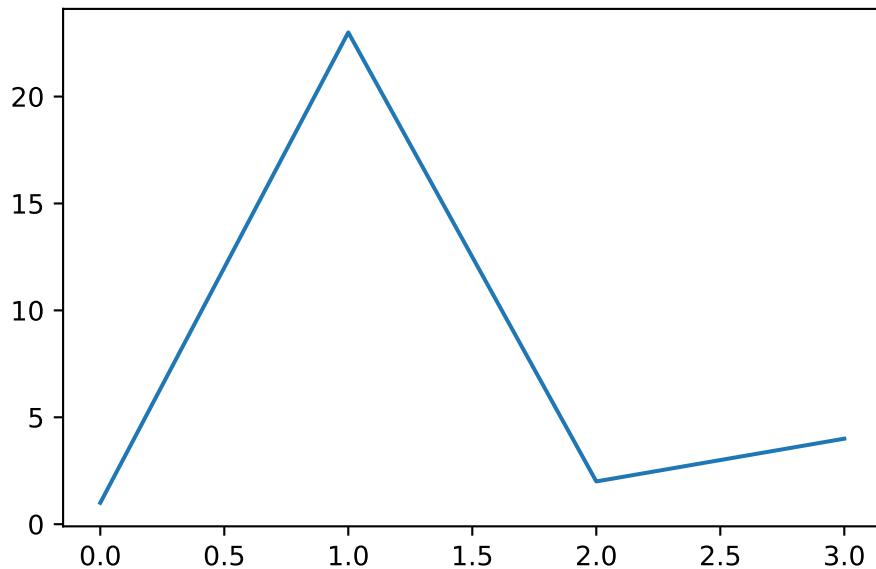


Figure 10.1: Simple Plot

In some cases, you may want to include a code-block that is not executed when the book is compiled. Use the `eval: false` option for this.

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

Figures can also be generated that do not show the code by using the option for `code-fold: true`.

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

r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
fig, ax = plt.subplots(
    subplot_kw = {'projection': 'polar'}
)
ax.plot(theta, r)
ax.set_rticks([0.5, 1, 1.5, 2])
ax.grid(True)
plt.show()
```

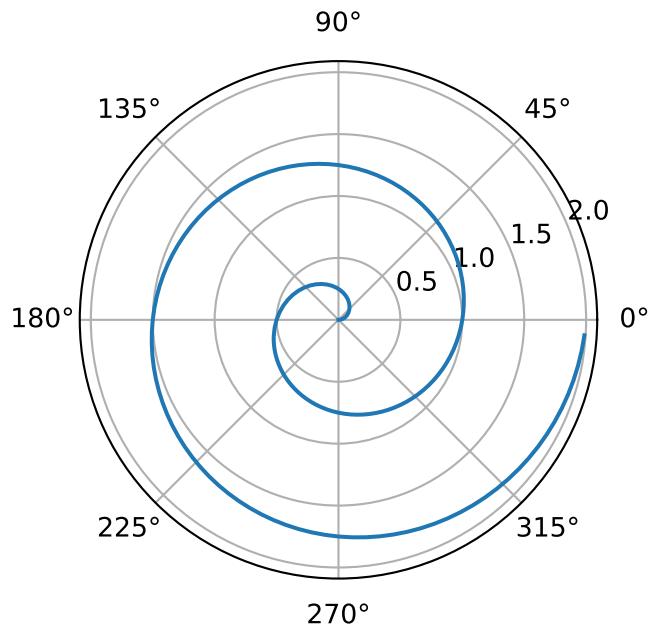


Figure 10.2: A spiral on a polar axis

Here is an example equation.

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (10.1)$$

10.4.1 Embedding Figures

You can also embed figures from other notebooks in the repo as shown in the following embed example.

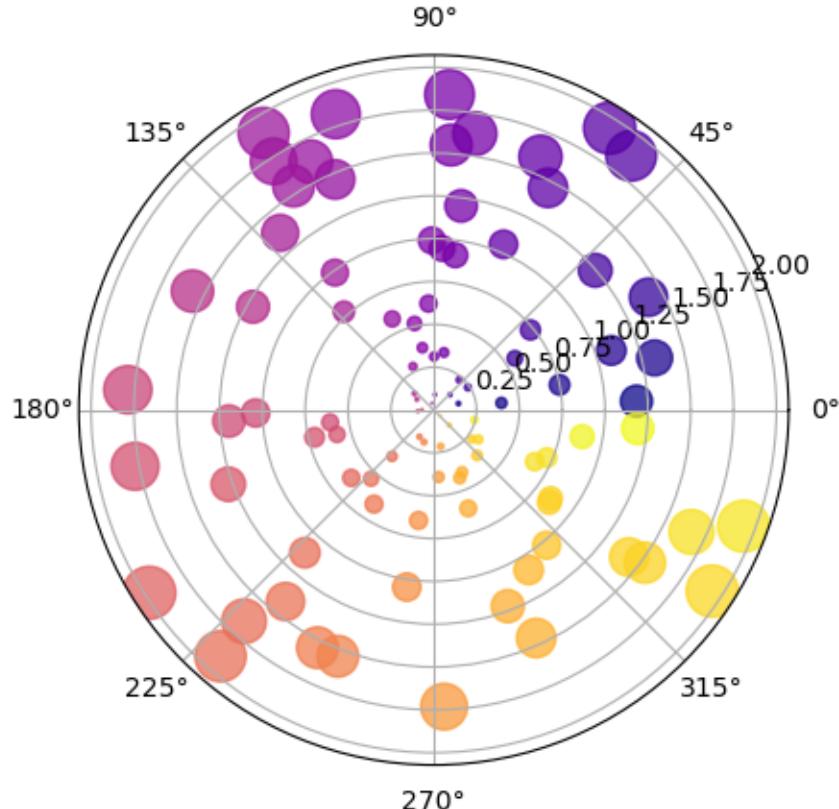


Figure 10.3: Polar plot of circles of random areas at random coords

When embedding notebooks, please store the .ipynb file in the notebook directory. Include the chapter in the name of your file. For example, `chapter4_example_u-net.ipynb`. This is how we will handle chapter- or example-specific environments. We will host notebooks on Google Colab so that any required packages for the code—but not for rendering the book at large—will be installed there. That way, we will not need to handle a global environment across the book.

10.5 Quarto has additional features.

You can learn more about markdown options and additional Quarto features in the [Quarto documentation](#). One example that you might find interesting is the option to include callouts in your text. These callouts can be used to highlight potential pitfalls or provide additional optional exercises that the reader might find helpful. Below are examples of the types of callouts available in Quarto.

Note

Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`. They can default to open (like this example) or collapsed (example below).

Tip

These could be good for extra material or exercises.

Caution

There are caveats when applying these tools. Expand the code below to learn more.

```
r = np.arange(0, 2, 0.01)  
theta = 2 * np.pi * r
```

Warning

Be careful to avoid hallucinations.

Important

This is key information.

11 Outlook

What can AI enable for biology?

Under your first header, include a brief introduction to your chapter.

Starting prompt for this chapter: Chapter 11 concludes the book with a forward-looking assessment of AI in microscopy, focusing on how it can/will enable biological discovery. It should highlight a few motivational examples of discoveries that AI has already enabled and discuss opportunities to which the reader is primed to contribute after reading this book.

11.1 Include section headers as appropriate

Use markdown heading level two for section headers. You can use standard markdown formatting, for example *emphasize the end of this sentence*.

This is a new paragraph with more text. Your paragraphs can cross reference other items, such as Figure 11.1. Use `fig` to reference figures, and `eq` to reference equations, such as Equation 11.1.

11.1.1 Sub-subsection headers are also available

To make your sections cross reference-able throughout the book, include a section reference, as shown in the header for Section 11.4.

11.2 Bibliography and Citations

To cite a research article, add it to references.bib and then refer to the citation key. For example, reference Stringer et al. [62] refers to CellPose and reference Chamier et al. [14] refers to ZeroCostDL4Mic.

11.3 Adding to the Glossary

We are using R code to create a glossary for this book. To add a definition, edit the glossary.yml file. To reference the glossary, enclose the word as in these examples: LLMs suffer from [hallucinations](#). It is important to understand the underlying [training data](#), [validation data](#) and [false positives](#) to interpret your results. Clicking on the word will reveal its definition by taking you to the entry on the [Glossary](#) page. Pressing back in your browser will return you to your previous place in the textbook.

11.4 Code and Equations

This is an example of including a python snippet that generates a figure

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

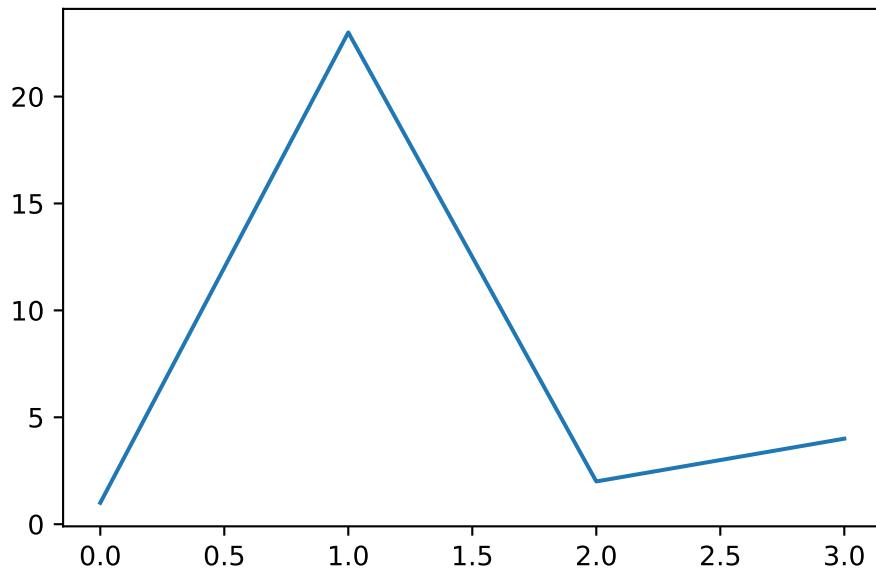


Figure 11.1: Simple Plot

In some cases, you may want to include a code-block that is not executed when the book is compiled. Use the `eval: false` option for this.

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

Figures can also be generated that do not show the code by using the option for `code-fold: true`.

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

r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
fig, ax = plt.subplots(
    subplot_kw = {'projection': 'polar'}
)
ax.plot(theta, r)
ax.set_rticks([0.5, 1, 1.5, 2])
ax.grid(True)
plt.show()
```

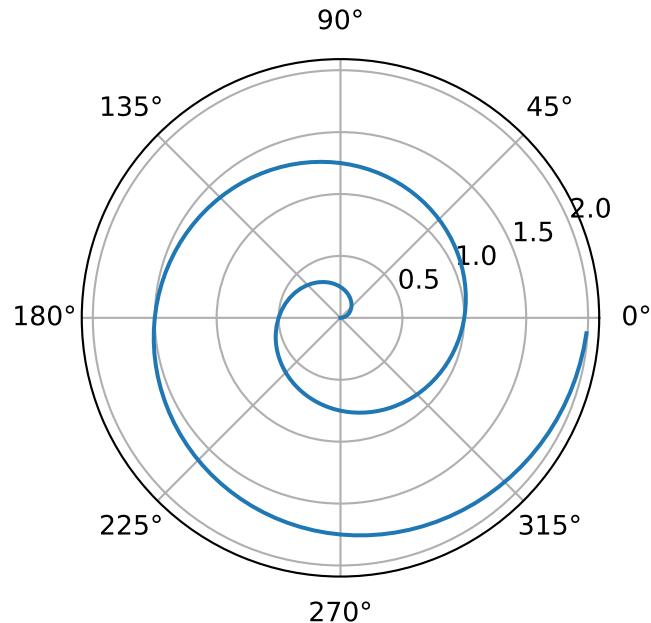


Figure 11.2: A spiral on a polar axis

Here is an example equation.

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (11.1)$$

11.4.1 Embedding Figures

You can also embed figures from other notebooks in the repo as shown in the following embed example.

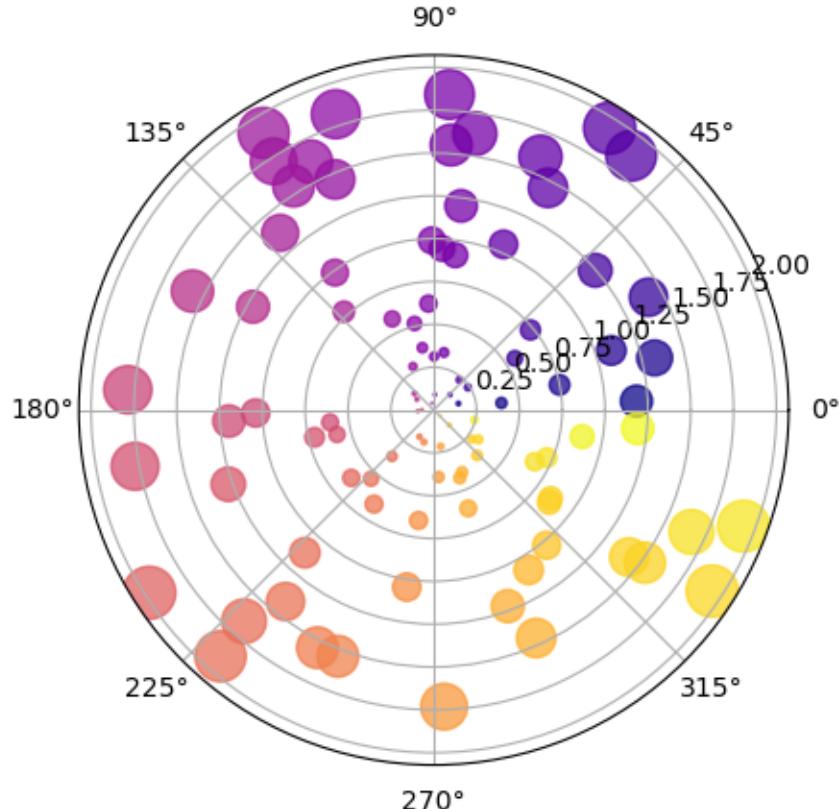


Figure 11.3: Polar plot of circles of random areas at random coords

When embedding notebooks, please store the .ipynb file in the notebook directory. Include the chapter in the name of your file. For example, `chapter4_example_u-net.ipynb`. This is how we will handle chapter- or example-specific environments. We will host notebooks on Google Colab so that any required packages for the code—but not for rendering the book at large—will be installed there. That way, we will not need to handle a global environment across the book.

11.5 Quarto has additional features.

You can learn more about markdown options and additional Quarto features in the [Quarto documentation](#). One example that you might find interesting is the option to include callouts in your text. These callouts can be used to highlight potential pitfalls or provide additional optional exercises that the reader might find helpful. Below are examples of the types of callouts available in Quarto.

Note

Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`. They can default to open (like this example) or collapsed (example below).

Tip

These could be good for extra material or exercises.

Caution

There are caveats when applying these tools. Expand the code below to learn more.

```
r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
```

Warning

Be careful to avoid hallucinations.

Important

This is key information.

Glossary

Backpropagation

The method used by neural networks to learn from its predictions. Once the prediction is done, it is compared with the ground truth through a training loss and the value of the comparison is used backwards to sequentially update the weights in the neural network, reward it when making a good prediction and punish it when making a bad prediction.

Batch

A small group of data that is processed together at the same time. For example, when training a machine learning model, a batch is a group of data that is given to the model for learning. Batches are commonly used to make the processes more efficient.

Bayesian Optimization

A strategy that allows the optimization of black-box functions such as deep neural networks. It creates a surrogate model, which is a probabilistic representation of the objective function, using only a few example points.

Binary Segmentation

A type of image segmentation where each pixel is classified into one of two categories—typically “foreground” (e.g., cell) or “background.” The output is a binary mask distinguishing objects (set to a value of 1) from their background (0).

Computer Vision

A field of computer science wherein computers extract information from images. It often involves object detection within images and can involve classification of the images and/or objects.

Data Augmentation

A strategy to artificially increase the diversity of a dataset prior to training by applying transformations such as rotation, flipping, or brightness adjustment. It helps improve model robustness and generalisation.

Domain Randomization

Using simulations or synthetic training data, domain randomization applies random and exaggerated variations to background, lighting, shapes, or textures in the synthetic dataset. This strategy helps the model learn domain-invariant features and is usually used for pretraining a neural network or to enable simulation-to-real transfer.

Effect Size

How “strong” a phenotype is, or how mathematically possible it is to distinguish a given population from the control population.

Epoch

One complete pass through the entire training dataset during the training process.

F1 Score

A classification metric that gives the harmonic mean of precision (proportion of correct true positive predictions across all predicted positive cases) and recall (proportion of true positive predictions against the total positive cases). The harmonic mean is a method to balance both metrics equally. This metric was originally designed for binary classification but can be adapted to multiclass classification by calculating the F1 score per class.

False Negatives

In a scenario where you have two classes “positive” and “negative”, you try to predict cases as one of those classes. False negatives are the cases that you incorrectly predicted as negative and were really positive.

False Positives

In a scenario where you have two classes “positive” and “negative”, you try to predict cases as one of those classes. False positives are the cases that you incorrectly predicted as positive and were really negative.

Gaussian Process

A common surrogate model for optimization strategies such as Bayesian Optimization. Gaussian Processes are non-parametric a case that models a conditional probability function. In the hyperparameter search scenario, the Gaussian Process models the probability of getting an objective function value based on some hyperparameters.

Genetic Algorithms

An optimisation method inspired by the principles of natural selection and genetics. It starts with a population of solutions. These solutions are combined through a process called crossover to produce new solutions (offspring). During this process, random changes or mutations may occur to introduce diversity. After crossover and mutation, a selection step chooses the best solutions from both the parent and offspring populations to form the next generation. This cycle repeats for a set number of generations or until a predefined goal or stopping criterion is met.

Ground Truth

Accurate data against which a model can be evaluated. Ground truth data is often manually annotated. The data type itself will vary depending on the task and evaluation. e.g. instance segmentation may be compared to ground truth object counts or masks.

Hallucinations

Outputs from a model that do not have a basis in the input data and may contain false or misleading information.

Hyperparameters

The options you choose when training a machine learning model that affect the training process or the architecture of the model (e.g., learning rate, batch size, number of layers, training loss, etc.) are called hyperparameters. This term is used to differentiate them from the parameters (also known as weights) of the machine learning model.

Image Classification

A computer vision task where each image is associated with one class and the goal of this task is to correctly predict that class.

Inferences

Temporary definition.

Instance Segmentation

A segmentation task that not only separates objects from the background but also distinguishes between individual objects of the same type (e.g., separating touching cells one by one).

IoU

“Intersection over Union”. A segmentation metric that calculates the difference between the area of overlap between two segmentation masks divided by the area of union.

Manual Annotation

The process of manually labeling specific structures or objects in an image using drawing tools. Typically done in software like Fiji or Napari, this step is essential for creating ground truth data to train or evaluate machine learning models.

Metadata

Any data that provides additional information about other data. In bioimaging, examples include information about sample preparation, the imaging instrument, and image acquisition parameters.

Object Detection

A computer vision task that identifies and locates individual objects within an image, typically by drawing bounding boxes around them. It provides both the category (what) and position (where) of each object.

Panoptic Segmentation

A computer vision technique that is a combination of semantic segmentation and instance segmentation. It separates an image into regions while also detecting individual object instances within those regions.

Pixel Classifiers

Machine learning models that classify each pixel in an image based on features such as intensity, texture, or local neighborhood. Commonly used in traditional workflows for segmentation or classification tasks.

Quality Control Metric

Any metric that can be used to evaluate quality. It will vary depending on the task and data type. It can be binary (e.g. an image doesn't have debris) or continuous (e.g. annotated object centroids are within 5 pixels of the ground truth centroids).

Semantic Segmentation

A form of segmentation where each pixel in an image is assigned to a class (e.g., nucleus, cytoplasm, background), but it does not distinguish between separate instances of the same class.

Star-convex Polygon

A geometric shape used in segmentation algorithms like StarDist. Imagine drawing straight lines (rays) from the centre of an object out toward its edges—if you can see the edge from the centre in all directions, the object is considered star-convex. This method works well for blob-like structures such as nuclei, because their general shape can be captured by measuring how far each ray travels from the centre to the boundary.

Test Data

Temporary definition.

Training Data

Data used to train an algorithm to make predictions.

Transfer Learning

A deep learning technique where part of a pretrained neural network (usually the initial layers, responsible for feature extraction) is frozen and reused in a new model. These frozen layers, with the knowledge from a previous dataset, are combined with untrained layers tailored for a specific bioimaging task. During training, only the new layers will be updated, allowing the model to adapt to the new task with limited data.

True Negatives

In a scenario where you have two classes “positive” and “negative”, you try to predict cases as one of those classes. True positives are the cases that you predicted as negative and were really negative.

True Positives

In a scenario where you have two classes “positive” and “negative”, you try to predict cases as one of those classes. True positives are the cases that you predicted as positive and were really positive.

Validation Data

Temporary definition.

Virtual Machine

On a physical computer, you install an operating system (e.g., Windows or Ubuntu) that you interact with. A virtual machine is a program that simulates a complete computer with its own operating system. This lets you run a “computer inside your computer” (e.g., using Linux inside Windows or the other way around). As this simulated computer is separate from your physical one, it adds an extra layer of security, because unless the user specifically allows it, the virtual machine cannot access or connect to your real computer.

References

- [1] Jannis Ahlers et al. *napari: a multi-dimensional image viewer for Python*. July 2023. DOI: [10.5281/zenodo.8115575](https://doi.org/10.5281/zenodo.8115575). URL: <https://zenodo.org/record/8115575> (visited on 10/05/2023).
- [2] Hussain Alibrahim and Simone A. Ludwig. “Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization”. In: *2021 IEEE Congress on Evolutionary Computation (CEC)*. 2021, pp. 1551–1559. DOI: [10.1109/CEC45853.2021.9504761](https://doi.org/10.1109/CEC45853.2021.9504761).
- [3] Anwai Archit et al. “Segment Anything for Microscopy”. en. In: *Nature Methods* 22.3 (Mar. 2025). Publisher: Nature Publishing Group, pp. 579–591. ISSN: 1548-7105. DOI: [10.1038/s41592-024-02580-4](https://doi.org/10.1038/s41592-024-02580-4). URL: <https://www.nature.com/articles/s41592-024-02580-4> (visited on 04/17/2025).
- [4] Anwai Archit et al. “Segment Anything for microscopy”. en. In: *Nat. Methods* 22.3 (Mar. 2025), pp. 579–591.
- [5] Ignacio Arganda-Carreras et al. “Trainable Weka Segmentation: a machine learning tool for microscopy pixel classification”. In: *Bioinformatics* 33.15 (Aug. 2017), pp. 2424–2426. ISSN: 1367-4803. DOI: [10.1093/bioinformatics/btx180](https://doi.org/10.1093/bioinformatics/btx180). URL: <https://doi.org/10.1093/bioinformatics/btx180> (visited on 08/22/2023).
- [6] Matthias Arzt et al. “LABKIT: Labeling and Segmentation Toolkit for Big Image Data”. In: *Frontiers in Computer Science* 4 (2022). ISSN: 2624-9898. URL: <https://doi.org/10.3389/fcomp.2022.777728> (visited on 08/30/2022).
- [7] Matthias Arzt et al. “LABKIT: Labeling and segmentation toolkit for big image data”. In: *Front. Comput. Sci.* 4 (Feb. 2022).
- [8] Peter Bankhead et al. “QuPath: Open source software for digital pathology image analysis”. en. In: *Sci. Rep.* 7.1 (Dec. 2017), p. 16878.
- [9] Peter Bankhead et al. “QuPath: Open source software for digital pathology image analysis”. en. In: *Scientific Reports* 7.1 (Dec. 2017). Number: 1 Publisher: Nature Publishing Group, p. 16878. ISSN: 2045-2322. DOI: [10.1038/s41598-017-17204-5](https://doi.org/10.1038/s41598-017-17204-5). URL: <https://doi.org/10.1038/s41598-017-17204-5> (visited on 10/11/2023).
- [10] Leire Bejarano et al. “Interrogation of endothelial and mural cells in brain metastasis reveals key immune-regulatory mechanisms”. In: *Cancer Cell* 42.3 (Mar. 2024), 378–395.e10. ISSN: 1535-6108. DOI: [10.1016/j.ccr.2023.12.018](https://doi.org/10.1016/j.ccr.2023.12.018). URL: <https://doi.org/10.1016/j.ccr.2023.12.018>. URL: <https://www.sciencedirect.com/science/article/pii/S1535610823004464> (visited on 04/22/2025).

- [11] Stuart Berg et al. “Ilastik: Interactive machine learning for (bio)image analysis”. en. In: *Nat. Methods* 16.12 (Dec. 2019), pp. 1226–1232.
- [12] Stuart Berg et al. “ilastik: interactive machine learning for (bio)image analysis”. en. In: *Nature Methods* 16.12 (Dec. 2019). Number: 12 Publisher: Nature Publishing Group, pp. 1226–1232. ISSN: 1548-7105. DOI: [10.1038/s41592-019-0582-9](https://doi.org/10.1038/s41592-019-0582-9). URL: <https://www.nature.com/articles/s41592-019-0582-9> (visited on 08/22/2023).
- [13] Juan C. Caicedo et al. “Weakly Supervised Learning of Single-Cell Feature Embeddings”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. ISSN: 2575-7075. June 2018, pp. 9309–9318. DOI: [10.1109/CVPR.2018.00970](https://doi.org/10.1109/CVPR.2018.00970). URL: <https://ieeexplore.ieee.org/document/8579068> (visited on 04/17/2025).
- [14] Lucas von Chamier et al. “Democratising deep learning for microscopy with Zero-CostDL4Mic”. en. In: *Nature Communications* 12.1 (Apr. 2021), p. 2276. ISSN: 2041-1723. DOI: [10.1038/s41467-021-22518-0](https://doi.org/10.1038/s41467-021-22518-0). URL: <https://www.nature.com/articles/s41467-021-22518-0> (visited on 06/24/2022).
- [15] Fabrice de Chaumont et al. “Icy: an open bioimage informatics platform for extended reproducible research”. en. In: *Nat. Methods* 9.7 (June 2012), pp. 690–696.
- [16] Jianxu Chen et al. *The Allen Cell and Structure Segmenter: a new open source toolkit for segmenting 3D intracellular structures in fluorescence microscopy images*. en. Pages: 491035 Section: New Results. Dec. 2020. DOI: [10.1101/491035](https://doi.org/10.1101/491035). URL: <https://www.biorxiv.org/content/10.1101/491035v2> (visited on 04/22/2025).
- [17] Beth A Cimini. *When to say 'good enough'*. Oct. 2019.
- [18] Ryan Conrad and Kedar Narayan. “Instance segmentation of mitochondria in electron microscopy images with a generalist deep learning model trained on a diverse dataset”. In: *Cell Systems* 14.1 (Jan. 2023), 58–71.e5. ISSN: 2405-4712. DOI: [10.1016/j.cels.2022.12.006](https://doi.org/10.1016/j.cels.2022.12.006). URL: <https://www.sciencedirect.com/science/article/pii/S240547122200494X> (visited on 04/22/2025).
- [19] Elnaz Fazeli et al. “Automated cell tracking using StarDist and TrackMate”. eng. In: *F1000Research* 9 (2020), p. 1279. ISSN: 2046-1402. DOI: [10.12688/f1000research.27019.1](https://doi.org/10.12688/f1000research.27019.1).
- [20] Daniel Fisch et al. “Molecular definition of the endogenous Toll-like receptor signalling pathways”. en. In: *Nature* 631.8021 (July 2024). Publisher: Nature Publishing Group, pp. 635–644. ISSN: 1476-4687. DOI: [10.1038/s41586-024-07614-7](https://doi.org/10.1038/s41586-024-07614-7). URL: <https://www.nature.com/articles/s41586-024-07614-7> (visited on 04/22/2025).
- [21] Gautier Follain et al. *Fast label-free live imaging reveals key roles of flow dynamics and CD44-HA interaction in cancer cell arrest on endothelial monolayers*. en. Pages: 2024.09.30.615654 Section: New Results. Oct. 2024. DOI: [10.1101/2024.09.30.615654](https://doi.org/10.1101/2024.09.30.615654). URL: <https://www.biorxiv.org/content/10.1101/2024.09.30.615654v1> (visited on 03/21/2025).
- [22] Thibaut Goldsborough et al. “InstanSeg: an embedding-based instance segmentation algorithm optimized for accurate, efficient and portable cell segmentation”. In: *arXiv [cs.CV]* (Aug. 2024).

- [23] Estibaliz Gómez-de-Mariscal et al. “DeepImageJ: A user-friendly environment to run deep learning models in ImageJ”. en. In: *Nat. Methods* 18.10 (Oct. 2021), pp. 1192–1195.
- [24] Robert Haase et al. “A call for FAIR and open-access training materials to advance BioImage Analysis”. In: (Mar. 2024).
- [25] Larissa Heinrich et al. “Whole-cell organelle segmentation in volume electron microscopy”. en. In: *Nature* 599.7883 (Nov. 2021). Publisher: Nature Publishing Group, pp. 141–146. ISSN: 1476-4687. DOI: [10.1038/s41586-021-03977-3](https://doi.org/10.1038/s41586-021-03977-3). URL: <https://www.nature.com/articles/s41586-021-03977-3> (visited on 04/17/2025).
- [26] Iván Hidalgo-Cenalmor et al. “DL4MicEverywhere: deep learning for microscopy made flexible, shareable and reproducible”. en. In: *Nature Methods* (May 2024). ISSN: 1548-7091, 1548-7105. DOI: [10.1038/s41592-024-02295-6](https://doi.org/10.1038/s41592-024-02295-6). URL: <https://www.nature.com/articles/s41592-024-02295-6> (visited on 05/20/2024).
- [27] Iván Hidalgo-Cenalmor et al. “DL4MicEverywhere: deep learning for microscopy made flexible, shareable and reproducible”. en. In: *Nat. Methods* 21.6 (June 2024), pp. 925–927.
- [28] Ilija Ilievski et al. “Efficient Hyperparameter Optimization for Deep Learning Algorithms Using Deterministic RBF Surrogates”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 31.1 (Feb. 2017). DOI: [10.1609/aaai.v31i1.10647](https://doi.org/10.1609/aaai.v31i1.10647). URL: <https://ojs.aaai.org/index.php/AAAI/article/view/10647>.
- [29] Nasim Jamali et al. “2020 BioImage Analysis Survey: Community experiences and needs for the future”. In: *Biological Imaging* 1 (Jan. 2021), e4.
- [30] Thomas Kluyver et al. “Jupyter Notebooks - a publishing format for reproducible computational workflows”. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by Fernando Loizides and Birgit Scmidt. Netherlands: IOS Press, 2016, pp. 87–90. URL: <https://eprints.soton.ac.uk/403913/>.
- [31] Bogdan Kochetov et al. “UNSEG: unsupervised segmentation of cells and their nuclei in complex tissue samples”. en. In: *Communications Biology* 7.1 (Aug. 2024). Publisher: Nature Publishing Group, pp. 1–14. ISSN: 2399-3642. DOI: [10.1038/s42003-024-06714-4](https://doi.org/10.1038/s42003-024-06714-4). URL: <https://www.nature.com/articles/s42003-024-06714-4> (visited on 04/22/2025).
- [32] Anna Kreshuk and Chong Zhang. “Machine Learning: Advanced Image Segmentation Using ilastik”. en. In: *Methods Mol. Biol.* 2040 (2019), pp. 449–463.
- [33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 10/05/2023).
- [34] Romain F. Laine et al. “Avoiding a replication crisis in deep-learning-based bioimage analysis”. In: *Nature methods* 18.10 (Oct. 2021), pp. 1136–1144. ISSN: 1548-7091. DOI: [10.1038/s41592-021-01284-3](https://doi.org/10.1038/s41592-021-01284-3). URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7611896/> (visited on 10/06/2023).

- [35] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: [10.1109/5.726791](https://doi.org/10.1109/5.726791). URL: <https://ieeexplore.ieee.org/document/726791> (visited on 04/22/2025).
- [36] Yuexiang Li and Linlin Shen. “cC-GAN: A Robust Transfer-Learning Framework for HEp-2 Specimen Image Segmentation”. In: *IEEE Access* 6 (2018), pp. 14048–14058. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2018.2808938](https://doi.org/10.1109/ACCESS.2018.2808938). URL: <https://ieeexplore.ieee.org/document/8301400> (visited on 04/17/2025).
- [37] Binbin Lin et al. “A deep learned nanowire segmentation model using synthetic data augmentation”. en. In: *npj Computational Materials* 8.1 (Apr. 2022). Publisher: Nature Publishing Group, pp. 1–12. ISSN: 2057-3960. DOI: [10.1038/s41524-022-00767-x](https://doi.org/10.1038/s41524-022-00767-x). URL: <https://www.nature.com/articles/s41524-022-00767-x> (visited on 04/22/2025).
- [38] Bojing Liu et al. “Self-supervised learning reveals clinically relevant histomorphological patterns for therapeutic strategies in colon cancer”. en. In: *Nature Communications* 16.1 (Mar. 2025). Publisher: Nature Publishing Group, p. 2328. ISSN: 2041-1723. DOI: [10.1038/s41467-025-57541-y](https://doi.org/10.1038/s41467-025-57541-y). URL: <https://www.nature.com/articles/s41467-025-57541-y> (visited on 04/17/2025).
- [39] Vebjorn Ljosa, Katherine L Sokolnicki, and Anne E Carpenter. “Annotated high-throughput microscopy image sets for validation”. en. In: *Nat. Methods* 9.7 (June 2012), p. 637.
- [40] Erick Moen et al. “Deep learning for cellular image analysis”. en. In: *Nature Methods* 16.12 (Dec. 2019). Number: 12 Publisher: Nature Publishing Group, pp. 1233–1246. ISSN: 1548-7105. DOI: [10.1038/s41592-019-0403-1](https://doi.org/10.1038/s41592-019-0403-1). URL: <https://www.nature.com/articles/s41592-019-0403-1> (visited on 11/30/2023).
- [41] Mohammad Amin Morid, Alireza Borjali, and Guilherme Del Fiol. “A scoping review of transfer learning research on medical image analysis using ImageNet”. In: *Computers in Biology and Medicine* 128 (Jan. 2021), p. 104115. ISSN: 0010-4825. DOI: [10.1016/j.compbiomed.2020.104115](https://doi.org/10.1016/j.compbiomed.2020.104115). URL: <https://www.sciencedirect.com/science/article/pii/S0010482520304467> (visited on 04/17/2025).
- [42] Nikita Moshkov et al. “Learning representations for image-based profiling of perturbations”. en. In: *Nature Communications* 15.1 (Feb. 2024). Publisher: Nature Publishing Group, p. 1594. ISSN: 2041-1723. DOI: [10.1038/s41467-024-45999-1](https://doi.org/10.1038/s41467-024-45999-1). URL: <https://www.nature.com/articles/s41467-024-45999-1> (visited on 04/17/2025).
- [43] *napari: a multi-dimensional image viewer for Python*. DOI: [10.5281/zenodo.3555620](https://doi.org/10.5281/zenodo.3555620).
- [44] Wei Ouyang et al. “BioImage Model Zoo: A Community-Driven Resource for Accessible Deep Learning in BioImage Analysis”. en. In: *bioRxiv* (June 2022), p. 2022.06.07.495102.
- [45] Wei Ouyang et al. “ImJoy: an open-source computational platform for the deep learning era”. en. In: *Nat. Methods* 16.12 (Dec. 2019), pp. 1199–1200.

- [46] Marius Pachitariu and Carsen Stringer. “Cellpose 2.0: how to train your own model”. en. In: *Nature Methods* (Nov. 2022). Publisher: Nature Publishing Group, pp. 1–8. ISSN: 1548-7105. DOI: [10.1038/s41592-022-01663-4](https://doi.org/10.1038/s41592-022-01663-4). URL: <http://www.nature.com/articles/s41592-022-01663-4> (visited on 11/09/2022).
- [47] Marius Pachitariu and Carsen Stringer. “Cellpose 2.0: how to train your own model”. en. In: *Nat. Methods* 19.12 (Dec. 2022), pp. 1634–1641.
- [48] Joanna W. Pylvänäinen et al. “Live-cell imaging in the deep learning era”. en. In: *Current Opinion in Cell Biology* 85 (Dec. 2023), p. 102271. ISSN: 09550674. DOI: [10.1016/j.ceb.2023.102271](https://doi.org/10.1016/j.ceb.2023.102271). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0955067423001205> (visited on 11/07/2023).
- [49] Luis Rangel DaCosta et al. “A robust synthetic data generation framework for machine learning in high-resolution transmission electron microscopy (HRTEM)”. en. In: *npj Computational Materials* 10.1 (July 2024). Publisher: Nature Publishing Group, pp. 1–11. ISSN: 2057-3960. DOI: [10.1038/s41524-024-01336-0](https://doi.org/10.1038/s41524-024-01336-0). URL: <https://www.nature.com/articles/s41524-024-01336-0> (visited on 04/22/2025).
- [50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. en. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: [10.1007/978-3-319-24574-4_28](https://doi.org/10.1007/978-3-319-24574-4_28).
- [51] Curtis T Rueden et al. “Scientific Community Image Forum: A discussion forum for scientific image software”. en. In: *PLoS Biol.* 17.6 (June 2019), e3000340.
- [52] Craig T. Russell et al. *bia-binder: A web-native cloud compute service for the bioimage analysis community*. arXiv:2411.12662 [q-bio]. Nov. 2024. DOI: [10.48550/arXiv.2411.12662](https://doi.org/10.48550/arXiv.2411.12662). URL: <http://arxiv.org/abs/2411.12662> (visited on 04/09/2025).
- [53] Johannes Schindelin et al. “Fiji: an open-source platform for biological-image analysis”. en. In: *Nat. Methods* 9.7 (June 2012), pp. 676–682.
- [54] Johannes Schindelin et al. “Fiji: an open-source platform for biological-image analysis”. en. In: *Nature Methods* 9.7 (2012). ISBN: 1548-7105, pp. 676–682. DOI: [10.1038/nmeth.2019](https://doi.org/10.1038/nmeth.2019). URL: [https://www.nature.com/articles/nmeth.2019](http://www.nature.com/articles/nmeth.2019).
- [55] Johannes Schindelin et al. “The ImageJ ecosystem: An open platform for biomedical image analysis”. en. In: *Mol. Reprod. Dev.* 82.7-8 (July 2015), pp. 518–529.
- [56] Uwe Schmidt et al. “Cell Detection with Star-Convex Polygons”. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*. Springer International Publishing, 2018, pp. 265–273.
- [57] Uwe Schmidt et al. “Cell Detection with Star-Convex Polygons”. en. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*. Ed. by Alejandro F. Frangi et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 265–273. ISBN: 978-3-030-00934-2. DOI: [10.1007/978-3-030-00934-2_30](https://doi.org/10.1007/978-3-030-00934-2_30).

- [58] Rajavi Shah, Nodar Gogoberidze, and Beth Cimini. *Bilayers*. URL: <https://github.com/bilayer-containers/bilayers>.
- [59] Connor Shorten and Taghi M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. In: *Journal of Big Data* 6.1 (July 2019), p. 60. ISSN: 2196-1115. DOI: [10.1186/s40537-019-0197-0](https://doi.org/10.1186/s40537-019-0197-0). URL: <https://doi.org/10.1186/s40537-019-0197-0> (visited on 04/22/2025).
- [60] Suganya Sivagurunathan et al. “Bridging imaging users to imaging analysis - A community survey”. en. In: *J. Microsc.* (Sept. 2023).
- [61] David R Stirling et al. “CellProfiler 4: improvements in speed, utility and usability”. en. In: *BMC Bioinformatics* 22.1 (Sept. 2021), p. 433.
- [62] Carsen Stringer et al. “Cellpose: a generalist algorithm for cellular segmentation”. en. In: *Nature Methods* 18.1 (Jan. 2021), pp. 100–106. ISSN: 1548-7105. DOI: [10.1038/s41592-020-01018-x](https://doi.org/10.1038/s41592-020-01018-x). URL: <https://www.nature.com/articles/s41592-020-01018-x> (visited on 06/03/2024).
- [63] Chong Zhang et al. “Bio-Image Informatics Index BIII: A unique database of image analysis tools and workflows for and by the bioimaging community”. In: *arXiv [q-bio.QM]* (Dec. 2023).