UNIVERSITÀ DEGLI STUDI DI CASSINO E DEL LAZIO MERIDIONALE

Corso di Laurea Magistrale in Ingegneria Informatica

Blood Cells Detection and Classification using Deep Learning and Machine Learning

Versione 0.1

Cannavale Achille
Colacicco Nunziamaria
La Torre Noemi

June 27, 2025

Contents

1 **3**

1.1 Introduzione . . . 3

1.2 Strumenti Utilizzati . . . 4

1.3 Composizione del Dataset . . . 4

2 Deep Learning **5**

2.1 Trasformazioni . . . 5

2.2 Defined Environments . . . 6

2.2.1 Writing Equations . . . 7

2.2.2 Designing a Table . . . 7

3 Plotting your data using PGF/TikZ **9**

3.1 Introduction . . . 9

3.1.1 A Simple 2D Plot . . . 10

3.1.2 Plotting 3D plots . . . 11

Chapter 1

Contents

1.1 Introduzione 3
1.2 Strumenti Utilizzati 4
1.3 Composizione del Dataset 4

1.1 Introduzione

La malaria è una delle principali malattie infettive a livello globale, con un impatto significativo sulla salute pubblica, in particolare nei paesi tropicali e subtropicali. Tra i diversi parassiti responsabili di questa patologia, il Plasmodium vivax rappresenta una delle specie più diffuse. La diagnosi precoce e accurata è fondamentale per un trattamento tempestivo ed efficace, e l'analisi microscopica degli strisci ematici rimane uno degli strumenti diagnostici più affidabili. In questo progetto, viene affrontato il problema della rilevazione e classificazione automatica delle cellule del sangue infettate da P. vivax, attraverso tecniche di machine e deep learning. Utilizzando il dataset P. vivax malaria infected human blood smears, che contiene oltre **1.300** immagini microscopiche annotate con più di **86.000** cellule identificate e classificate, si propone una pipeline (Figura 1.1) che combina **Deep Learning** e **Machine Learning** per:

- individuare automaticamente le regioni di interesse (ROI) contenenti cellule
- estrarre le feature significative dalle ROI
- classificare ogni cellula in una delle categorie predefinite, tra cui cellule sane (red blood cell e leucocity) e diversi stadi del parassita infetto (trophozoite, ring, gametocyte, ecc.)

L'analisi di questo lavoro ha evidenziato una marcata sbilanciatura del dataset, aspetto che ha richiesto un'attenta progettazione delle strategie di addestramento.

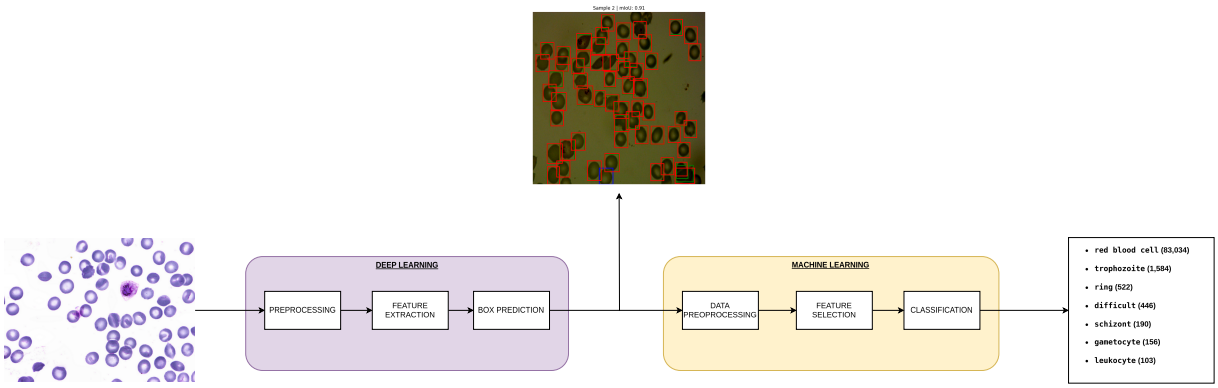

Figure 1.1: Pipeline Generale del nostro progetto, suddivisa in una fase di Deep Learning per per la rilevazione delle ROI e l'estrazione delle features e una fase di Machine Learning per la classificazione delle cellule.

1.2 Strumenti Utilizzati

Per lo sviluppo del progetto, sono stati utilizzati i seguenti strumenti software:

- Parte di **Deep Learning:**
  - **PyTorch**: per la progettazione, addestramento e valutazione dei modelli di Deep Learning
- Parte di **Machine Learning:**
- **Scikit-Learn**: per l'implementazione di algoritmi di Machine Learning
- Strumenti **Ausiliari:**
  - **Numpy** e **Pandas**: per l'analisi dei dati
  - **Matplotlib** e **Seaborn**: per la visualizzazione dei dati e dei risultati

1.3 Composizione del Dataset

Il Dataset preso in esame è composto da 1.328 immagini e da 86.035 oggetti. Ogni oggetto può appartenere ad una delle seguenti 7 classi:

- **red blood cell** (RBC): globuli rossi sani (83,034)
- **trophozoite**: stadio iniziale del parassita (1,584)
- **ring**: stadio di anello del parassita (1,617)
- **difficult**: stadio difficile del parassita (446)
- **schizont**: stadio avanzato del parassita (190)
- **gametocyte**: stadio gametico del parassita (156)
- **leukocyte**: globuli bianchi sani (103)

Da qui possiamo già evincere il grandissimo squilibrio presente nel dataset, rappresentato dalla classe maggioritaria delle **red blood cell**.

Chapter 2

Deep Learning

Contents

2.1 Trasformazioni 5
2.2 Defined Environments 6
2.2.1 Writing Equations 7
2.2.2 Designing a Table 7

2.1 Trasformazioni

Per garantire una buona generalizzazione del modello, sono state provate diverse trasformazioni, usando la libreria Albumentation, tra le quali sono state scelte le seguenti migliori per il nostro caso di studio: (anche una tabella va bene)

Trasformazione	Parametri
A.HorizontalFlip	p=0.5
A.VerticalFlip	p=0.5
A.ColorJitter	brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1, p=0.3
A.RandomBrightnessContrast	p=0.3
A.MotionBlur	p=0.2
A.GaussNoise	p=0.2
A.CLAHE	p=0.2
A.CoarseDropout	num holes ange=(3, 6), hole height range=(10, 20), hole width range=(10, 20), fill="random uniform", p=0.2
A.Resize	800x800
A.Normalize	mean=[0.7205, 0.7203, 0.7649], std=[0.2195, 0.2277, 0.1588]

Table 2.1: Lista delle trasformazioni utilizzate per il dataset.

In particolare, I parametri della normalizzazione sono stati calcolati dal dataset stesso:

Parametro	Valore
Media RGB	(0.7205, 0.7203, 0.7649)
Deviazione standard RGB	(0.2195, 0.2277, 0.1588)

Table 2.2: Valori di media e deviazione standard per la normalizzazione delle immagini.

Una delle trasformazioni che hanno garantito un forte miglioramento è rappresentata da CoarseDropout, che randomicamente oscura regioni rettangolari dall'immagine, simulando l'occlusione ottica e variando la grandezza degli oggetti nel mondo reale. (esempi immagine)

2.2 Defined Environments

`excerpt` The template relies on the excellent `tcolorbox` package for formatting the boxes within the document and for that end different styles were created.

Sometimes one needs to quote either a proverb or to create drama, for this use the `excerpt` environment with the following notation and effect.

```
\begin{excerpt}
  To be, or not to be...
\end{excerpt}
```

Compiling this code snippet would show as in the document

> To be, or not to be...

`code` During the preparation of your document, it is useful to showcase some code either in the shape of all the document or a snippet of it. There are two (2) ways of doing this where the first one will be discussed here.

For example to print out a hello world in python, please use the following environment

```
\begin{code}{python}
print("Hello, World!")
\end{code}
```

Producing the following:

```python
print("Hello, World!")
```

The class also come with some predefined environments to modify the behaviour/aesthetics of the document. Highlighting text is very easy, here is an example on how to write one.

```
Highlighting text is \hlight{very easy}, here is an example:
```

`example` Sometimes you need to showcase an example or need to highlight a certain idea. For these things the environment Example could be useful.

For example to show as simple example or give a slight attention to a topic you can do the following.

> This is an example. This could be anything which you would like to have a certain amount of attention but not too much as to distract from the flow of the document.

`highlight` Or sometimes you need to give a clear break to the flow of the document and ask the reader to look at your banner. For that use highlight.

> Hey! Pay attention as this is a highlight box.

2.2.1 Writing Equations

One of the strong suits of LaTeX compared to other editors and programs is it simplicity and ease of use methods of writing equations. Consider the following equation:

$$f(x) = x^2 + 2x + 1$$

In code form this would be written as:

```
\begin{equation*}
        f(x) = x^2 + 2x + 1
\end{equation*}
```

All equations that has their newline and centre staged are mostly written in an environment where it has a `begin` and an `end`. You may have noticed the asterisks sign just after the equation. This implies the environment is not numbered, meaning you won't be able to reference it. This is used to limit the numbering of equations to just the essential parts in the document and not reach 3 digits by the time you are in page 8. For a numbered equation like the following

$$f(x) = x^2 + 2x + 1 \tag{2.1}$$

You only need to do:

```
\begin{equation}\label{eq:quad}
        f(x) = x^2 + 2x + 1
\end{equation}
```

where `\label{eq:quad}` is the equation reference label. You could also make matrices as well as `amsmath` is preloaded into this template.

2.2.2 Designing a Table

Finally, no template is done without someone telling you how a table should be designed. Below is a standard table And the code used to generate it:

```
\begin{table}[!ht]
        \begin{NiceTabular}{rX}[rules/color=[gray]{0.9},rules/width=1pt]
                \CodeBefore
                \rowcolors{1}{black!5}{}
```

Section	Scientific Method Step
Introduction	states your hypothesis
Methods	details how you tested your hypothesis
Results	provides raw (i.e., uninterpreted) data collected
Discussion	considers whether the data you obtained support the hypothesis

Table 2.3: A Detailed look into the scientific method.

```
                \rowcolors{3}{blue!5}{}
                \Body
                \toprule
                \textbf{Section}      & \textbf{Scientific Method Step}    \\
                \midrule
                \textbf{Introduction} & states   hypothesis                \\
                \textbf{Methods}      & how you tested hypothesis          \\
                \textbf{Results}      & provides raw  data collected       \\
                \textbf{Discussion}   &  whether it support the hypothesis \\
                \bottomrule
        \end{NiceTabular}
        \caption{A Detailed look into the scientific method.}
\end{table}
```

Chapter 3

Plotting your data using PGF/TikZ

Contents

3.1 Introduction 9
3.1.1 A Simple 2D Plot 10
3.1.2 Plotting 3D plots 11

3.1 Introduction

PGFplots and Tikz are powerful scripting languages allowing you to draw high-quality diagrams using only a programming language. PGFplots are generally used for plotting data from a wide variety of representations from simple 2D plots to complex 3D geometries.
But wikipedia description put it best:

> PGF/TikZ is a pair of languages for producing vector graphics (e.g., technical illustrations and drawings) from a geometric/algebraic description, with standard features including the drawing of points, lines, arrows, paths, circles, ellipses and polygons. PGF is a lower-level language, while TikZ is a set of higher-level macros that use PGF. The top-level PGF and TikZ commands are invoked as TeX macros, but in contrast with PSTricks, the PGF/TikZ graphics themselves are described in a language that resembles MetaPost.

For more info please look at the documentation here. It is of course up to the user to select which graphical software to produce the necessary visual components but unless it requires complex functions/processing, it would be be easier to have it in PGF/TikZ format for easy editing/maintenance.
For this manual we will be looking at the three (3) plot types you may encounter in your studies.

3.1.1 A Simple 2D Plot

2D plots are simple yet powerful to show the relation of a single parameters and its related function. Below is an example of a simple comparison of two (2) functions. The image above

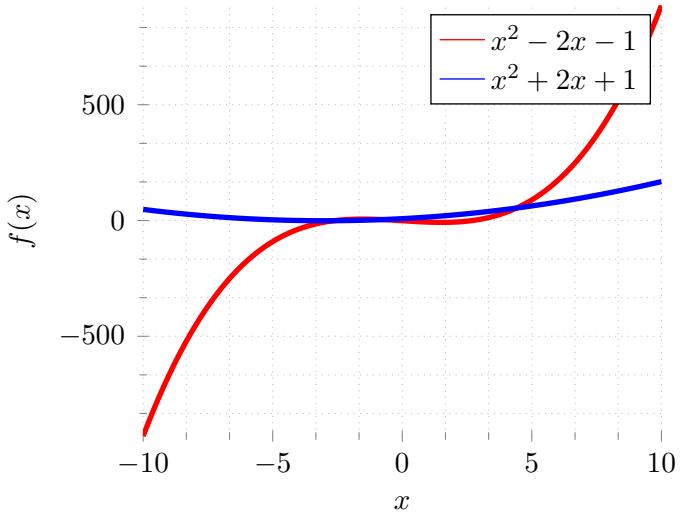

Figure 3.1: This is an example of a 2D PGF plot comparing two functions where these functions are calculated using PGF itself rather than entering/reading from data.

is generated using the following code:

```
\begin{figure}[!ht]
  \centering
  \begin{tikzpicture}
    \begin{axis}[hebdomon, xlabel = \(x\), ylabel = {\(f(x)\)}]
      %
      \addplot [domain=-10:10, samples=100,red]{x^3 - 7*x - 1};
      \addlegendentry{\(x^2 - 2x - 1\)}
      %
      \addplot [domain=-10:10, samples=100, blue]{x^2 + 6*x + 8};
      %
      \addlegendentry{\(x^2 + 2x + 1\)}
      %
    \end{axis}
  \end{tikzpicture}
  \caption{This is an example of a 2D PGF plot comparing
  two functions where these functions are calculated using
  PGF itself rather than entering/reading from data.}
\end{figure}
```

As can be seen it is relatively standard to create plots. Some aspect which need mentioning.

`\addplot` You invoke this command when you want to create a plot. In the square brackets (i.e., []) you insert your configuration of your plot. The most important ones are

`domain` the range in which the function will be calculated

`sample` the number of calculations will be done within the defined domain.

3.1.2 Plotting 3D plots

Plotting data with PGFplots is also quite possible and will generate great plot (as long as it is not massively complicated). For more information on the precautions on designing 3D plots, please have a look at here.
Below is the prototypical plot to showcase the 3D capabilities of PGF:

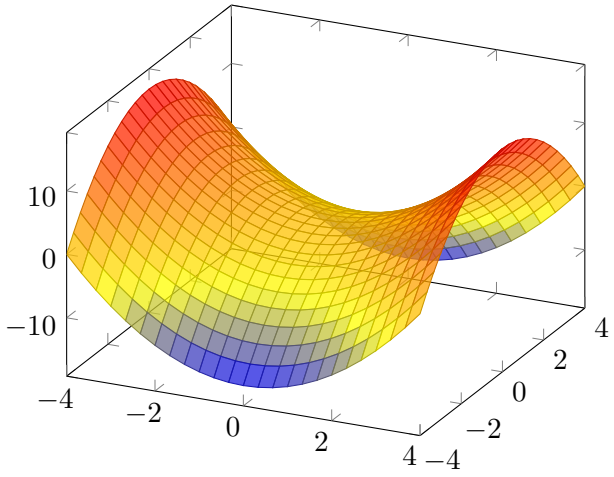

Figure 3.2: An example 3D plot done wit PGFplots.

And, of course the code for generating the plot is given as follows:

```
\begin{figure}[!ht]
  \centering
  \begin{tikzpicture}
    \begin{axis}[view={25}{30},mark layer=like plot]
      \addplot3 [
      surf,
      shader=faceted,
      fill opacity=0.75,
      samples=25,
      domain=-4:4,
      y domain=-4:4,
      on layer=main,
      ] {x^2-y^2};
    \end{axis}
  \end{tikzpicture}
  \caption{An example 3D plot done wit PGFplots.}
\end{figure}
```

Some options worth mentioning are as follows:

`surf` Generates a surface based on the 2D data it was given (in this case these are x and y.

`shader` Describes, basically how each segment should be filled.

`samples` Similar to 2D plots, tells how many data points will be measured. However, make a note that 3D is significantly more taxing on the TeX memory than 2D and making this sampling high may result in exceeding the memory limit.