



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

**Cannavale Achille
Colacicco Nunziamaria
La Torre Noemi**

Indice

1		3
1.1	Introduzione	3
1.2	Strumenti utilizzati	4
1.3	Composizione Dataset	4
2	Deep Learning	6
2.1	Introduzione	6
2.2	Trasformazioni	7
2.3	Personalizzazione delle Anchor Boxes	8
2.4	Focal Loss	8
2.5	Architetture del Modello	9
2.6	Valutazione	11
2.6.1	Iperparametri	11
2.6.2	Visualizzazione dei Risultati	12
2.7	Estrazione delle Feature	13
3	Machine Learning	15
3.1	Introduzione	15
3.2	Preparazione del Dataset	18
3.3	Oversampling: BorderlineSMOTE	18
3.4	Variance Threshold	19
3.5	Normalizzazione	19
3.6	Feature Selection: SelectKBest	19
3.7	Dimensionality Reduction: LDA	19
3.8	Approccio Gerarchico alla Classificazione	20
3.9	Model training	21
3.10	Model evaluation	22

Capitolo 1

Contents

1.1	Introduzione	3
1.2	Strumenti utilizzati	4
1.3	Composizione 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, in 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 **leukocyte**) e diversi stadi del parassita infetto (**trophozoite**, **ring**, **gametocyte**, **schizont**)

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

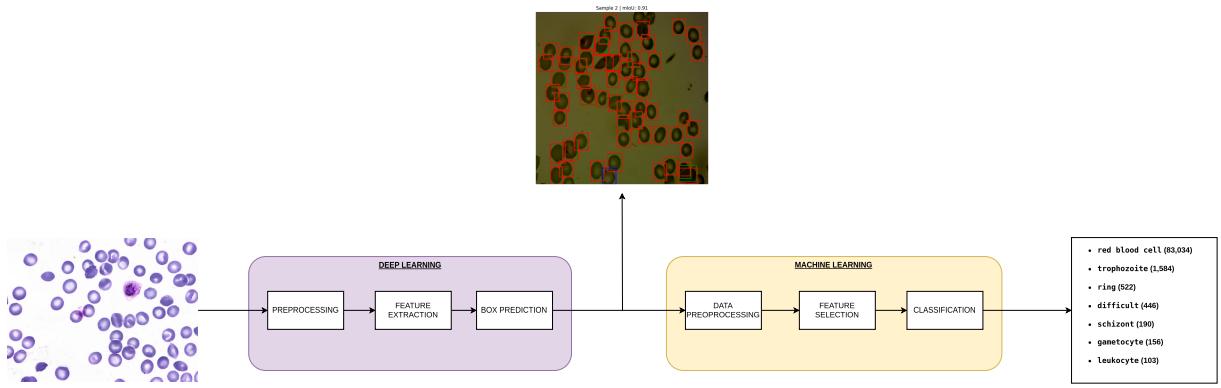


Figura 1.1: Pipeline Generale del nostro progetto, suddivisa in una fase di Deep Learning per la rilevazione delle ROI e l'estrazione delle features e una fase di Machine Learning per la classificazione delle cellule.

1.2 Strumenti utilizzati

Per lo sviluppo del progetto sono stati adottati diversi strumenti software, suddivisi in base alle funzionalità specifiche per il *deep learning*, il *machine learning* e l'analisi dei dati. La scelta di ciascuno di essi è stata guidata da criteri di efficienza, semplicità d'uso e ampia diffusione nella comunità scientifica.

- **Deep Learning:** *PyTorch*

Utilizzato per la progettazione, l'addestramento e la valutazione dei modelli di deep learning. PyTorch si distingue per la sua flessibilità e l'approccio dinamico alla costruzione delle reti neurali, risultando particolarmente adatto per attività di ricerca e prototipazione rapida.

- **Machine Learning:** *Scikit-learn*

Impiegato per l'implementazione di algoritmi di machine learning tradizionali, grazie alla sua vasta collezione di modelli predefiniti e strumenti per la valutazione delle prestazioni.

- **Strumenti Ausiliari:**

- *NumPy* e *Pandas*: utilizzati per la manipolazione, la pulizia e l'analisi dei dati, fondamentali nella fase di preprocessing e gestione dei dataset.
- *Matplotlib*: utilizzata per la visualizzazione grafica dei risultati, utile per l'analisi esplorativa e la presentazione delle performance dei modelli.

1.3 Composizione Dataset

Il dataset utilizzato per questo progetto è composto da **1.328 immagini**, con una risoluzione di (1600 x 1200) contenenti in totale **86.035 oggetti annotati**. Ogni oggetto rappresenta una singola cellula o elemento biologico rilevato nelle immagini microscopiche, ed è associato a una delle seguenti **sette classi**:

- red blood cell (83.034)

- trophozoite (1.584)
- ring (522)
- difficult (446)
- schizont (190)
- gametocyte (156)
- leukocyte (103)

Come si può osservare dalla distribuzione, il dataset presenta un **forte sbilanciamento tra le classi**, con la stragrande maggioranza degli oggetti appartenenti alla categoria **red blood cell**, che rappresenta da sola circa il **96,5%** del totale.

Questo sbilanciamento costituisce una **sfida rilevante per i modelli di classificazione**, in quanto tende a favorire la previsione della classe dominante a discapito delle classi minoritarie, che possono essere trascurate o predette con bassa accuratezza.

Per affrontare questo problema, durante lo sviluppo sono state considerate diverse strategie, tra cui:

- il bilanciamento del dataset,
- l'uso di tecniche di *data augmentation*,
- l'assegnazione di pesi differenti alle classi durante l'addestramento,
- e l'adozione di metriche di valutazione più robuste rispetto alla sola accuratezza, come *precision*, *recall* e *F1-score*.

Capitolo 2

Deep Learning

Contents

2.1	Introduzione	6
2.2	Trasformazioni	7
2.3	Personalizzazione delle Anchor Boxes	8
2.4	Focal Loss	8
2.5	Architetture del Modello	9
2.6	Valutazione	11
2.6.1	Iperparametri	11
2.6.2	Visualizzazione dei Risultati	12
2.7	Estrazione delle Feature	13

2.1 Introduzione

Come primo passo preliminare allo sviluppo del sistema completo, è stato effettuato un test esplorativo utilizzando il modello **YOLO Ultralytics 8** (You Only Look Once), una delle architetture più diffuse per il rilevamento e la classificazione di oggetti in immagini. L'obiettivo di questa prova iniziale era quello di valutare la complessità del dataset e ottenere un'indicazione generale sulla difficoltà del compito di classificazione delle cellule, prima di procedere con pipeline più specializzate.

YOLO è stato scelto in quanto permette una rapida sperimentazione ed è noto per la sua capacità di eseguire inferenza in tempo reale su immagini contenenti più oggetti, rendendolo un buon punto di partenza per una valutazione qualitativa del problema. Il dataset, costituito da immagini delle cellule e relative annotazioni, è stato quindi adattato al formato richiesto da YOLO e sottoposto a training utilizzando una configurazione standard.

Il risultato ottenuto, un **mAP@50** pari a **0.39**, ha confermato che il problema presenta un certo grado di complessità. Questo valore, relativamente basso rispetto a quelli comunemente

ottenuti su dataset più strutturati, suggerisce che le classi sono tra loro visivamente molto simili e difficilmente distinguibili solo attraverso un approccio diretto di object detection.

2.2 Trasformazioni

Per garantire una buona generalizzazione del modello, e ridurre l'overfitting, è stata utilizzata la tecnica di *data augmentation* tramite la libreria Albumentations. Dopo una fase di sperimentazione, sono state selezionate le seguenti trasformazioni come le più efficaci per il nostro caso di studio:

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]

Tabella 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)

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

Una delle trasformazioni che ha avuto il maggiore impatto sulle prestazioni è stata la CoarseDropout (fig. 2.1), che oscura regioni rettangolari casuali dell'immagine, simulando occlusioni ottiche e migliorando la robustezza del modello in scenari reali dove gli oggetti potrebbero essere parzialmente visibili.

```
A.CoarseDropout(
    num_holes_range=(3, 6),
    hole_height_range=(10, 20),
    hole_width_range=(10, 20),
    fill="random_uniform",
        # Numero casuale di buchi tra 3 e 6
        # Altezza dei buchi tra 10 e 20 px
        # Larghezza dei buchi tra 10 e 20 px
        # Riempie i buchi con valori casuali
```

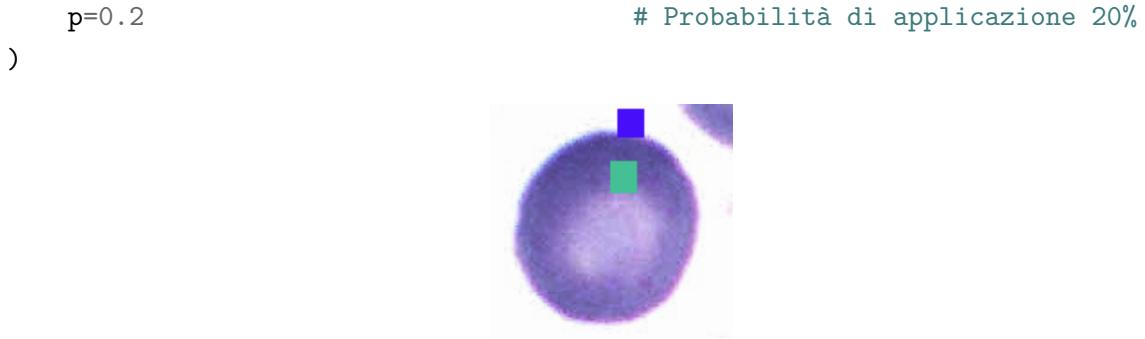


Figura 2.1: Esempio di CoarseDropout applicata su una cellula.

2.3 Personalizzazione delle Anchor Boxes

Per migliorare le prestazioni del modello di object detection, abbiamo effettuato un'analisi approfondita della distribuzione delle dimensioni delle bounding box presenti nel dataset. Le anchor boxes predefinite, come quelle utilizzate nei dataset generici come COCO, non risultavano ottimali per il nostro caso specifico, in quanto le cellule da rilevare presentano dimensioni e proporzioni piuttosto omogenee e differenti rispetto agli oggetti comuni.

A tal fine, abbiamo estratto tutte le bounding box del dataset di addestramento e calcolato le dimensioni (larghezza e altezza) di ciascuna. I dati così ottenuti sono stati utilizzati come input per un algoritmo di clustering **K-Means**, impostato per individuare i centroidi rappresentativi delle dimensioni più ricorrenti. In particolare, è stato scelto un numero di cluster pari a 6, corrispondente al numero di anchor boxes da definire. Le nuove ancore suggerite sono state quindi utilizzate per sostituire quelle standard all'interno del modello.

Questa ottimizzazione ha permesso un miglior allineamento tra le proposte del modello e le reali dimensioni degli oggetti nel dataset, contribuendo a un incremento consistente delle metriche di accuratezza. L'impatto dell'adozione delle nuove anchor boxes è illustrato nella Tabella 2.3, dove si evidenzia un miglioramento del valore di mAP rispetto alla configurazione predefinita.

Configurazione Anchor	mAP@0.5
Default (COCO-style)	0.75
Personalizzate (K-Means)	0.78

Tabella 2.3: Confronto tra ancore standard e personalizzate.

2.4 Focal Loss

Un aspetto cruciale di questo problema è il forte sbilanciamento del dataset: le cellule sane (globuli rossi e leucociti) sono molto più numerose rispetto alle cellule infette, che rappresentano una minoranza spesso difficile da distinguere.

Questo squilibrio comporta che durante l'addestramento i modelli tendano a privilegiare la classificazione corretta delle classi maggioritarie, a discapito di quelle minoritarie, che sono però di massima importanza per una diagnosi accurata e precoce della malaria.

Per mitigare questo problema, abbiamo adottato la *Focal Loss* (Lin et al., 2017), una funzione di perdita che estende la Cross Entropy introducendo un termine di modulazione. Tale termine riduce il peso degli esempi facili (correttamente classificati) e concentra l'apprendimento sugli esempi difficili o mal classificati, tipicamente appartenenti alle classi minoritarie. La formulazione della Focal Loss è:

$$FL(p_t) = -\alpha(1 - p_t)^\gamma \log(p_t) \quad 2.2$$

dove p_t è la probabilità stimata per la classe corretta, α è un fattore di bilanciamento tra classi e γ , detto "focusing parameter", controlla la riduzione del peso degli esempi facili.

Nel nostro caso, i parametri $\alpha = 0.25$ e $\gamma = 3.0$ sono stati scelti per ottenere un buon equilibrio tra stabilità e miglioramento delle performance.

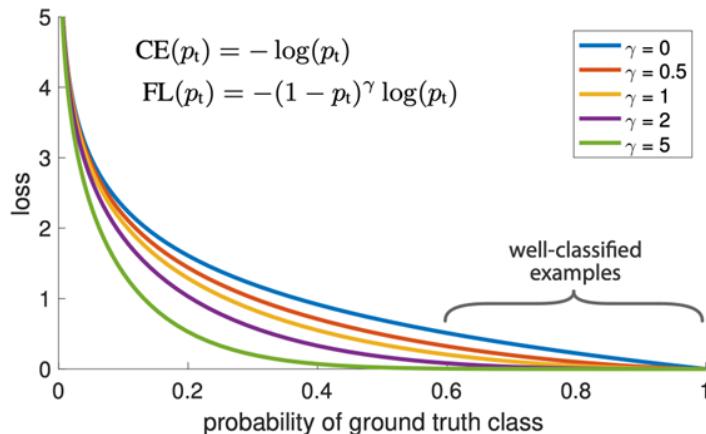


Figura 2.2: Grafico Focal Loss.

2.5 Architetture del Modello

Durante lo sviluppo del progetto sono state testate diverse architetture di rete neurale, combinando vari **backbone** con due principali tipologie di **head**: **Faster R-CNN** e **RetinaNet**. L'obiettivo era identificare la combinazione più efficace in termini di accuratezza, capacità di generalizzazione e prestazioni computazionali.

In un modello di object detection, il **backbone** funge da estrattore di feature: riceve l'immagine in input e produce mappe di caratteristiche utili per individuare e classificare gli oggetti. I backbone testati nel progetto includono:

- **ResNet50**: una rete convoluzionale profonda basata su **residual blocks**, che permette un'ottima propagazione del gradiente anche in reti molto profonde.

- **ResNet101**: una versione più profonda di ResNet50, che migliora la rappresentazione delle feature a scapito di un maggiore costo computazionale.
- **MobileNet**: una rete più leggera ed efficiente, pensata per dispositivi mobili, ma con performance inferiori in contesti complessi.
- **ResNeXt101**: un'estensione di ResNet che introduce la dimensione della cardinalità (cioè il numero di percorsi paralleli in ogni blocco), migliorando la capacità di rappresentazione senza aumentare drasticamente i parametri.

Il **detection head**, invece, riceve le feature estratte e produce le predizioni finali (classi e bounding box). Le due architetture di head utilizzate sono:

- **Faster R-CNN**: un approccio a due stadi, in cui un **Region Proposal Network (RPN)** genera proposte di oggetti che vengono successivamente classificate e regolarizzate. È noto per l'elevata accuratezza, ma è più lento rispetto ad approcci monostadio.
- **RetinaNet**: un metodo **one-stage** che introduce la **Focal Loss** per gestire lo squilibrio tra oggetti e background, migliorando le performance su oggetti piccoli o rari. Offre un buon compromesso tra accuratezza e velocità.

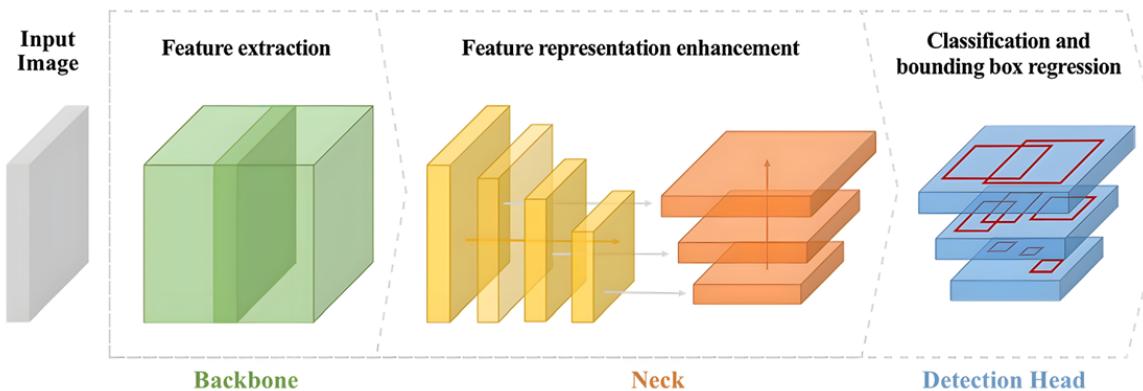


Figura 2.3: Pipeline Generale di una rete.

Le configurazioni testate sono le seguenti:

- **Backbone:** ResNet50 **Head:** Faster R-CNN
- **Backbone:** ResNet101 **Head:** Faster R-CNN
- **Backbone:** MobileNet **Head:** Faster R-CNN
- **Backbone:** ResNet50 **Head:** RetinaNet
- **Backbone:** ResNeXt101 **Head:** Faster R-CNN
- **Backbone:** ResNet101 **Head:** RetinaNet

L'architettura che ha fornito i migliori risultati, sia in termini di accuratezza sia di generalizzazione, è stata quella composta da **ResNeXt101** come backbone e **Faster R-CNN** come head. In Figura 2.4 è riportata una rappresentazione schematica della pipeline del modello.

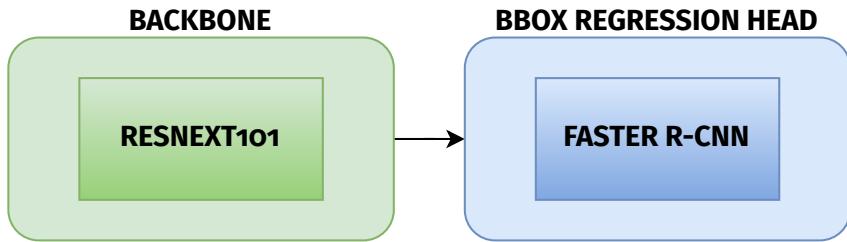


Figura 2.4: Pipeline Generale del miglior modello, composto da ResNeXt101 come backbone e Faster R-CNN come head.

2.6 Valutazione

2.6.1 Iperparametri

Parameter	Value
BATCH_SIZE	4
NUM_EPOCHS	5
LEARNING_RATE	0.0003
WEIGHT_DECAY	0.001
PATIENCE	5
OPTIM	AdamW
SCHEDULER	ReduceLROnPlateau
CustomAnchor	True
loss	FocalLoss
VAL_SIZE	0.2

Tabella 2.4: Configurazione degli Iperparametri del Miglior Modello.

Gli iperparametri riportati nella Tabella 2.4 sono il risultato di un'attenta fase di sperimentazione in cui sono stati confrontati diversi valori e configurazioni. In particolare, si è osservato che un numero di epoche pari a 5 è sufficiente per ottenere una buona convergenza: aumentare il numero di epoche a 10 non ha portato a miglioramenti significativi in termini di accuratezza o generalizzazione.

Per quanto riguarda l'ottimizzatore, si è confrontato l'uso di **Adam** e **AdamW**, con quest'ultimo che ha dimostrato maggiore stabilità e performance migliori, probabilmente grazie alla gestione più efficace della regolarizzazione tramite **weight decay**.

La politica di gestione del learning rate è stata valutata tra **ReduceLROnPlateau** e **CosineAnnealingLR**. **ReduceLROnPlateau** si è rivelata più adatta al nostro task, poiché riduceva il learning rate solo quando la metrica di validazione non migliorava, consentendo un adattamento più dinamico e mirato rispetto al cosine annealing, che segue un programma fisso.

Infine, la scelta della funzione di perdita si è concentrata tra la **standard loss** e la **Focal Loss**. Quest'ultima ha mostrato un vantaggio concreto nella gestione dello sbilanciamento delle classi presenti nel dataset, migliorando la capacità del modello di rilevare correttamente anche le classi meno rappresentate. Nella tabella 2.5 sono riportate le metriche di valutazione ottenute con questa configurazione ottimizzata.

Metric	Value
mIoU	0.8871
Precision	0.9733
Recall	0.9683
F1 Score	0.9708
mAP	0.7397
mAP@50	0.9479
mAP@75	0.8874

Tabella 2.5: Evaluation metrics of ResNeXt101 + Faster R-CNN model.

2.6.2 Visualizzazione dei Risultati

In questa sezione vengono mostrati i risultati ottenuti dall'addestramento del modello. La Figura 2.5 presenta il grafico dell'andamento della **training loss** e della **validation loss** durante le epoch, evidenziando una buona convergenza e un'assenza di overfitting significativo.

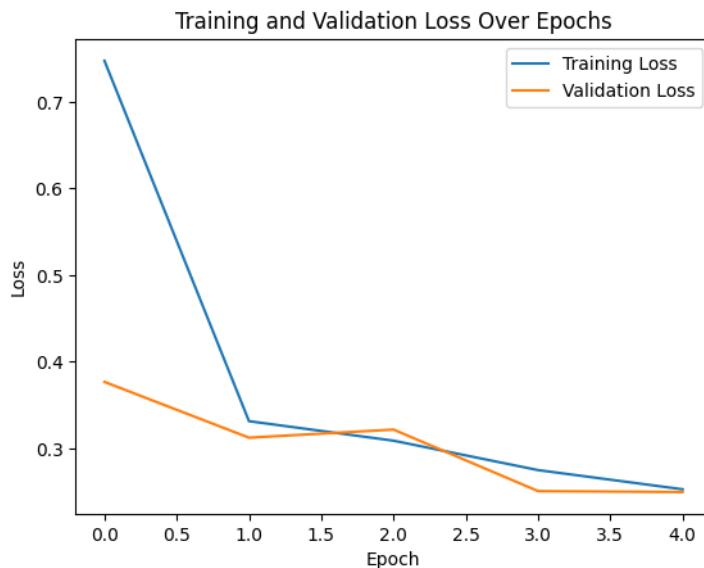


Figura 2.5: Grafico della Training Loss e della Validation Loss.

La Figura 2.6 mostra un esempio di output del modello su un'immagine del set di test. Le regioni di interesse (ROI) delle cellule sono rappresentate tramite rettangoli di diversi colori: le ROI esatte del dataset sono in **verde**, quelle predette dal modello in **rosso**, mentre in **blu** sono evidenziate le cellule non rilevate. Il modello raggiunge un valore medio di **mIoU** pari a **0.88**, confermando un'alta precisione nella localizzazione degli oggetti, con un'ottima corrispondenza tra predizioni e annotazioni di riferimento.

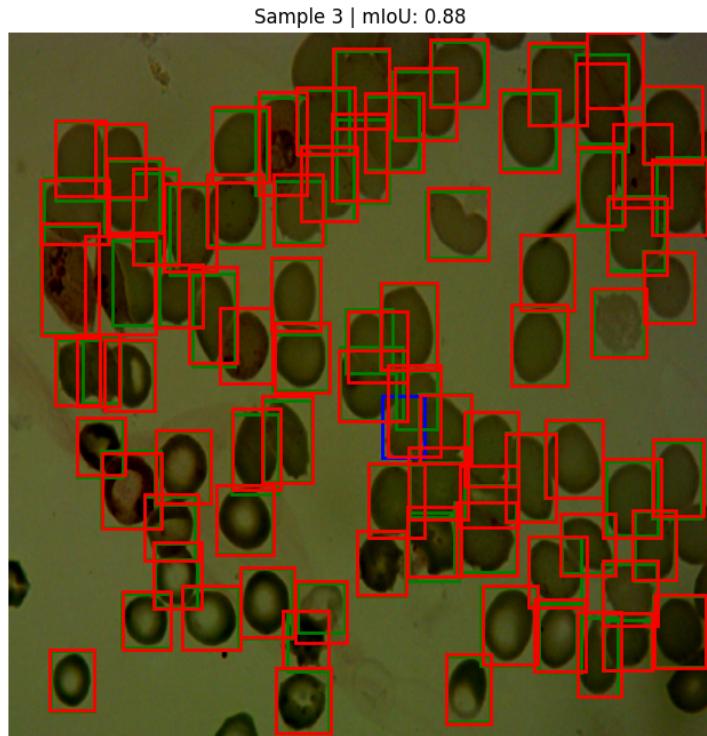


Figura 2.6: Risultato di esempio. In rosso sono evidenziate le bounding box correttamente previste dal modello, in blu quelle mancate.

2.7 Estrazione delle Feature

Per l'analisi delle rappresentazioni apprese dal modello, abbiamo implementato una procedura di estrazione delle caratteristiche visive associate alle ROIs predette. Il metodo si basa su un modello di object detection preaddestrato, nello specifico una variante di **Faster R-CNN con Feature Pyramid Network (FPN)**, che consente di sfruttare rappresentazioni multi-scala per migliorare la localizzazione e classificazione di oggetti di dimensioni variabili.

Durante l'inferenza, ogni immagine del dataset viene elaborata per ottenere le predizioni del modello, comprensive di bounding boxes e punteggi di confidenza. Le predizioni con uno score inferiore a `score_threshold` (default: 0.5) vengono scartate. Le restanti predizioni vengono confrontate con i ground truth tramite l'Intersection over Union (IoU), mantenendo solo quelle che superano una soglia di similarità. Questo matching consente di associare ciascuna ROI predetta alla corrispondente annotazione reale.

Le feature vengono estratte dai livelli intermedi del modello tramite la **backbone con FPN**, che restituisce una gerarchia di mappe di attivazione a risoluzioni diverse. A partire da queste mappe, il modulo `box_roi_pool` seleziona e aggrega le porzioni rilevanti in base alle bounding boxes predette. Le feature pooled vengono poi elaborate dalla `box_head`, e ulteriormente arricchite concatenando i **logit del classificatore** (`cls_score`), ottenendo così una rappresentazione congiunta che riflette sia le informazioni visive locali sia le risposte semantiche del modello.

Ogni vettore di feature viene associato a metadati esplicativi, tra cui:

- l'identificativo dell'immagine
- le coordinate della bounding box predetta
- il punteggio di confidenza
- l'etichetta del ground truth associato

Tutti i dati raccolti vengono infine aggregati in un dataframe strutturato, che unisce metadati e rappresentazioni numeriche. Il risultato viene salvato in formato CSV per facilitarne l'analisi quantitativa e la visualizzazione.

Capitolo 3

Machine Learning

Contents

3.1	Introduzione	15
3.2	Preparazione del Dataset	18
3.3	Oversampling: BorderlineSMOTE	18
3.4	Variance Threshold	19
3.5	Normalizzazione	19
3.6	Feature Selection: SelectKBest	19
3.7	Dimensionality Reduction: LDA	19
3.8	Approccio Gerarchico alla Classificazione	20
3.9	Model training	21
3.10	Model evaluation	22

3.1 Introduzione

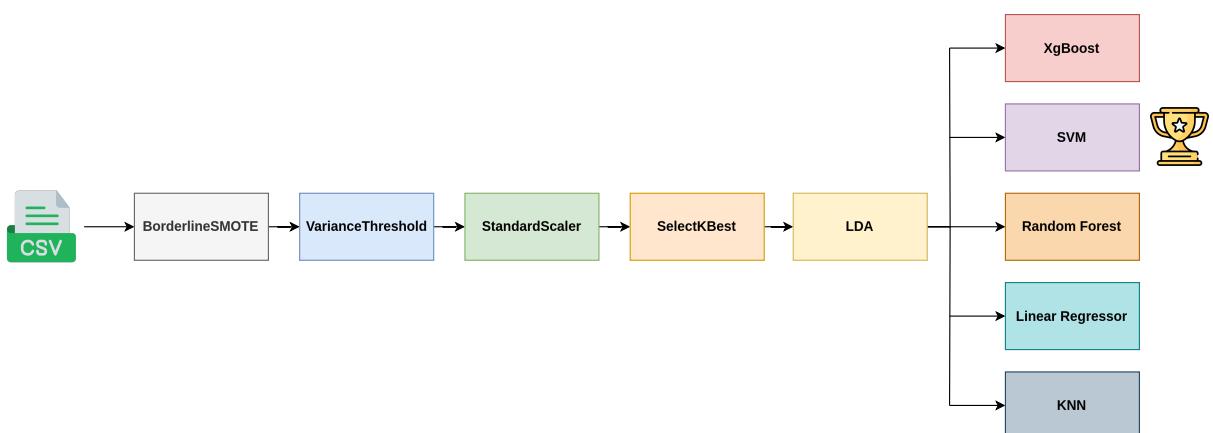


Figura 3.1: Pipeline generale della parte di Machine Learning.

La parte di Machine Learning ha come input il file csv generato nella parte di Deep Learning, che contiene le feature estratte dalle ROIs.

Inizialmente abbiamo tentato di classificare tutte le 7 classi contemporaneamente, utilizzando tecniche standard per l'analisi di dataset sbilanciati (come è possibile evincere dalla figura 3.4), come **SMOTE** e **PCA** per la riduzione dimensionale. Tuttavia, le classi risultavano difficilmente separabili nello spazio delle feature: la **PCA** mostrava un forte accavallamento tra le classi, con particolare predominanza dei globuli rossi (RBC), i quali rappresentavano la maggioranza nel dataset. Questo sbilanciamento si rifletteva anche nelle metriche di performance: i RBC raggiungevano un **F1-score del 90%**, mentre le altre classi ottenevano valori molto bassi, indicando una classificazione inefficace.

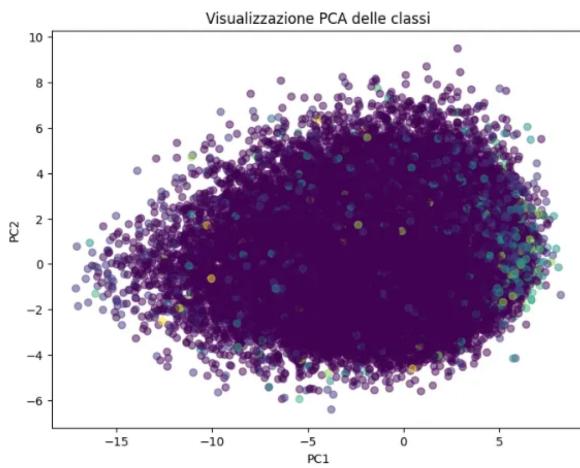


Figura 3.2: Risultato classificazione con PCA.

Per comprendere meglio la distribuzione delle sole cellule infette, abbiamo temporaneamente escluso le cellule sane dal training, filtrando manualmente le etichette, e sostituito la PCA con la tecnica **SelectKBest** per la selezione delle feature. Questo approccio ha portato a un significativo miglioramento delle performance: utilizzando un classificatore **SVM** si è raggiunto un **F1-score dell'89%**, da come è possibile evincere in figura 3.3, che mostra la distribuzione delle classi delle cellule infette dopo l'applicazione di LDA (Linear Discriminant Analysis) in `n_components=5`.

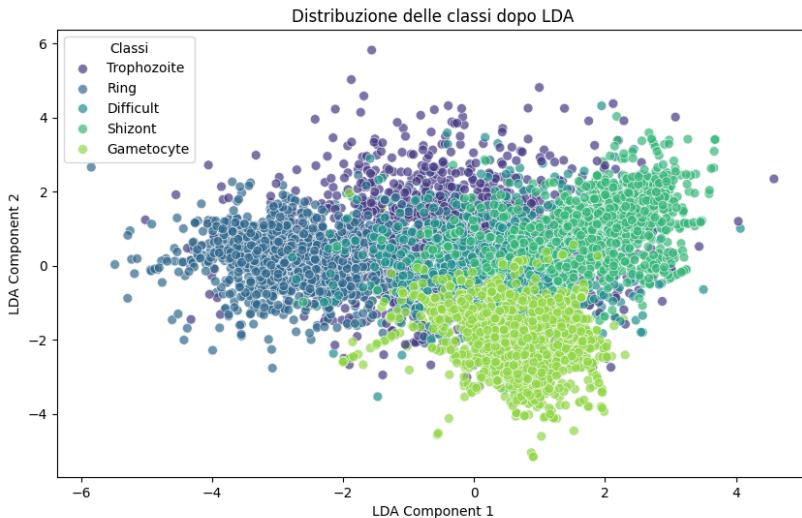


Figura 3.3: Distribuzione delle classi delle Cellule Infette.

L'introduzione della validazione incrociata (k -fold) ha causato un leggero calo delle performance, ma l'applicazione di una variante più sofisticata di SMOTE, il borderline-SMOTE, ha contribuito a migliorare nuovamente i risultati.

Durante l'analisi esplorativa, è emerso che la distinzione tra globuli rossi (RBC) e globuli bianchi (leukocyte) era particolarmente netta, raggiungendo un **F1-score del 99%** in un setting binario limitato a queste due classi.

Questo ha suggerito un nuovo approccio: suddividere il problema in due fasi distinte. La prima fase consiste in una classificazione binaria che separa le **cellule sane** (RBC + leukocyte) da quelle **infette** (tutte le altre). In questa configurazione, utilizzando LDA (Linear Discriminant Analysis) in 1D, è stato possibile ottenere una buona separabilità tra i gruppi, con un **F1-score macro del 93%** usando **Random Forest**.

Successivamente, si è provato a costruire un classificatore a 3 classi:

- RBC
- leukocyte
- cellule infette

Ma le performance si sono abbassate significativamente, con un **F1-score medio del 71%**.

Questa osservazione ha portato all'adozione di un **approccio a cascata**, composto da due stadi:

- **Classificatore 1 – Cellule sane vs Cellule Infette:** distingue le cellule sane da quelle infette.
- **Classificatore 2 – Cellule Infette Classification:** classifica solo le cellule infette nelle rispettive cinque sottoclassi.

Questo approccio modulare ha permesso di affrontare in maniera più efficace il problema dello sbilanciamento e della sovrapposizione tra classi, migliorando la robustezza e l'accuratezza complessiva del sistema di classificazione.

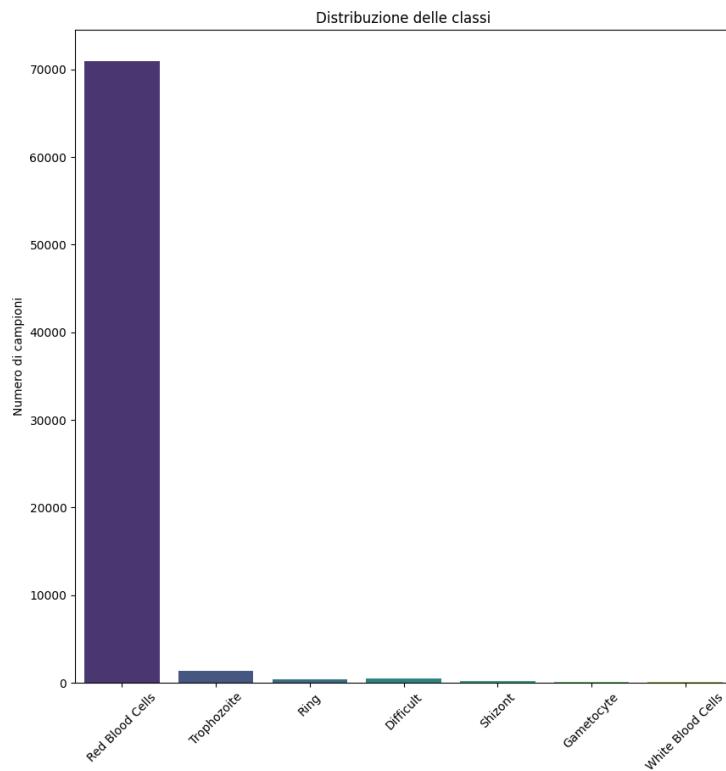


Figura 3.4: Distribuzione delle classi nel dataset prima del preprocessing.

3.2 Preparazione del Dataset

Il dataset non presenta valori mancanti, quindi non è stato necessario effettuare operazioni di **data cleaning**. Le etichette sono state codificate utilizzando un **LabelEncoder**, trasformando le classi da etichette testuali o intere non sequenziali in valori numerici consecutivi, come richiesto dalla maggior parte dei modelli di classificazione supervisionata.

3.3 Oversampling: BorderlineSMOTE

Per affrontare il problema dello sbilanciamento delle classi, è stato utilizzato l'algoritmo **BorderlineSMOTE** (in figura 3.5), una variante del Synthetic Minority Over-sampling Technique (SMOTE). Questo metodo genera nuovi campioni sintetici per le classi minoritare, migliorando la capacità del modello di apprendere le caratteristiche distintive delle classi meno rappresentate.

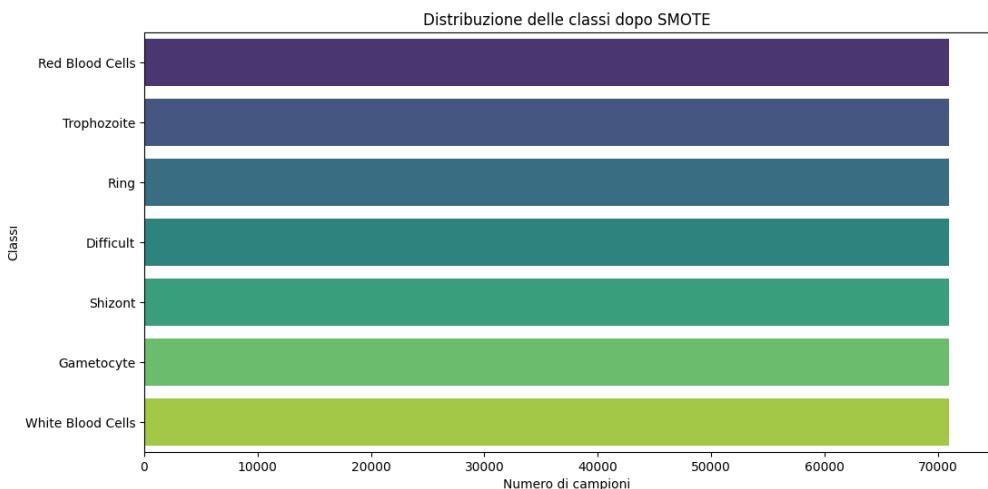


Figura 3.5: Distribuzione delle classi nel dataset dopo BorderlineSMOTE.

3.4 Variance Threshold

Per ridurre la dimensionalità del dataset e migliorare le prestazioni del modello, è stata applicata la tecnica di **Variance Threshold**. Questa tecnica rimuove le feature con una varianza inferiore a una soglia specificata, eliminando le caratteristiche che non apportano informazioni significative. Nel nostro caso, è stata impostata una soglia di `1e-5`, portando le features da **1024 a 767**.

3.5 Normalizzazione

Poiché le feature presentano range e scale molto differenti, è stata applicata una **standardizzazione** tramite **StandardScaler**, che porta ogni variabile ad avere media nulla e deviazione standard unitaria. Questo step è fondamentale per tutti gli algoritmi sensibili alla scala delle feature (come SVM, LDA, KNN), in quanto garantisce pari rilevanza a tutte le variabili numeriche.

3.6 Feature Selection: SelectKBest

Per contenere la dimensionalità del dataset e ridurre l'overfitting, è stata effettuata una selezione automatica delle feature tramite il metodo **SelectKBest**, basato su test ANOVA F-test (`f_classif`). Questo metodo assegna un punteggio a ogni feature in base alla sua correlazione con la variabile target, selezionando le k migliori.

3.7 Dimensionality Reduction: LDA

A valle della selezione, è stata applicata anche una tecnica di **riduzione dimensionale supervisata** tramite **Linear Discriminant Analysis** (LDA), che proietta i dati in uno spazio a dimen-

sionalità ridotta (fino a $n_{classi} - 1$) massimizzando la separabilità tra le classi. Questo aiuta il modello a concentrarsi solo sulle direzioni rilevanti per la classificazione.

LDA si basa su un principio statistico: cerca di massimizzare il rapporto tra la **varianza inter-classe** e la **varianza intra-classe**. Formalmente, l'obiettivo è trovare una proiezione W tale che:

$$W = \arg \max_W \frac{|W^T S_B W|}{|W^T S_W W|}$$

dove:

- S_B è la matrice di **scatter inter-classe**, che misura la distanza tra i centri delle varie classi.
- S_W è la matrice di **scatter intra-classe**, che misura la dispersione dei dati all'interno delle stesse classi.

Il risultato è uno spazio trasformato in cui le classi sono, idealmente, più separabili. Inoltre, il numero massimo di componenti ottenibili con LDA è pari a $C - 1$, dove C è il numero di classi.

3.8 Approccio Gerarchico alla Classificazione

Al fine di affrontare in modo più efficace lo sbilanciamento del dataset e migliorare la precisione della classificazione, si è scelto di adottare un **approccio gerarchico** suddiviso in due stadi:

- **Primo stadio – Classificazione Sano vs Infetto:**

Il primo classificatore ha il compito di distinguere tra cellule sane (globuli rossi e globuli bianchi) e cellule infette. Questa distinzione è particolarmente rilevante per evitare confusione tra classi visivamente simili ma semanticamente diverse. Per aumentare la separabilità tra le due categorie, è stata applicata la **Linear Discriminant Analysis (LDA)** come tecnica di riduzione dimensionale in uno spazio monodimensionale (1D), massimizzando la distanza tra le due classi target.

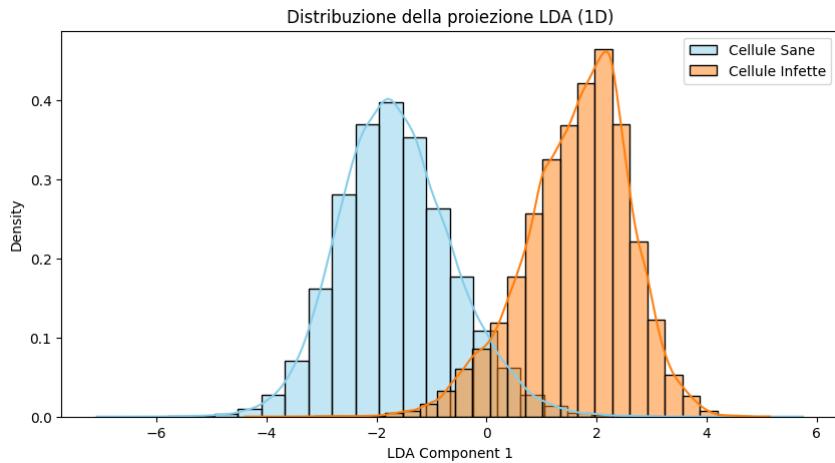


Figura 3.6: Distribuzione delle classi Cellule Sane e Cellule Infette.

Come mostrato in Figura 3.6, l'LDA consente di ottenere una buona separazione tra cellule sane e infette lungo una singola componente discriminante, facilitando il compito del classificatore.

- **Secondo stadio – Classificazione delle cellule infette:**

Una volta identificate le cellule infette, esse vengono passate a un secondo classificatore, il cui obiettivo è quello di distinguere tra le sottoclassi patologiche: trophozoite, ring, difficult, schizont e gametocyte. In questo caso, il numero massimo di componenti LDA applicabili è $n_{classi} - 1 = 5$. È stata quindi eseguita un'ulteriore riduzione dimensionale tramite LDA, al fine di valutare la separabilità delle classi anche in questo sottoinsieme.

Questo supporta l'efficacia dell'approccio gerarchico, in quanto la prima distinzione grossolana permette di alleggerire il carico del secondo classificatore, migliorando la performance complessiva del sistema.

3.9 Model training

Sono stati valutati diversi modelli di classificazione, tra cui:

- **Non parametrici:** K-Nearest Neighbors (KNN)
- **Lineari:** Logistic Regression
- **Support Vector Machine (SVM):** SVM con kernel lineare, polinomiale e RBF
- **Classificatori d'insieme:** Random Forest, XGBoost

dove:

- **K-Nearest Neighbors (KNN):** È un algoritmo non parametrico basato sulla distanza. Per classificare un'istanza, considera i k esempi più vicini (in genere in termini di distanza euclidea) e assegna la classe più rappresentata tra questi. KNN non assume alcuna

forma funzionale dei dati e può adattarsi bene a problemi non lineari, ma può essere sensibile alla scelta di k e alle caratteristiche dei dati (come la scala delle feature).

- **Logistic Regression:** È un modello lineare di classificazione che stima la probabilità che un'osservazione appartenga a una certa classe utilizzando la funzione logistica (sigmoide). È semplice, interpretabile ed efficace quando le classi sono linearmente separabili, ma può avere prestazioni limitate su problemi complessi e non lineari.
- **Support Vector Machine (SVM):** L'SVM cerca l'iperpiano che massimizza il margine tra le classi. Grazie all'uso del kernel trick, può proiettare i dati in spazi di dimensioni superiori, permettendo la separazione anche in casi non linearmente separabili. È efficace in spazi ad alta dimensionalità ma può richiedere una buona scelta di iperparametri (come C e il tipo di kernel).
- **Random Forest:** È un metodo ensemble basato su una collezione di alberi decisionali addestrati su diversi sottocampioni del dataset. La classificazione viene effettuata tramite voto di maggioranza. È robusto, gestisce bene l'overfitting e può modellare relazioni complesse, ma tende ad essere meno interpretabile.
- **XGBoost (Extreme Gradient Boosting):** È un potente algoritmo di boosting che costruisce alberi in sequenza, dove ciascun nuovo albero corregge gli errori dei precedenti. Ottimizza una funzione di perdita tramite gradient boosting ed è noto per la sua alta accuratezza e efficienza computazionale. Tuttavia, richiede una maggiore attenzione nell'ottimizzazione degli iperparametri per evitare overfitting.

L'ottimizzazione degli iperparametri è stata effettuata mediante `RandomizedSearchCV`, con una validazione incrociata a **3 fold** e ottimizzazione rispetto all'**F1 score macro**, al fine di tenere conto dell'eventuale sbilanciamento tra le classi.

3.10 Model evaluation

Al termine della fase di addestramento, il miglior modello di classificazione trovato si è rivelato essere **SVM** in entrambi gli stage, con la corrispettiva miglior configurazione degli iperparametri, trovata utilizzando `RandomizedSearchCV`:

```
param_dist = {
    'selector__k': [700],
    'classifier__C': [100],
    'classifier__kernel': ['poly'],
    'classifier__gamma': ['auto']
}
```

La valutazione delle performance del modello finale è stata effettuata confrontando le predizioni sul test set con le etichette reali, utilizzando le seguenti metriche:

- **Balanced Accuracy:** particolarmente utile in presenza di classi sbilanciate.

- **F1-Score Macro:** media armonica tra precision e recall, calcolata su tutte le classi, trattandole equamente.
- **Precision e Recall per classe**, per una valutazione più granulare delle performance.
- **Matrice di confusione**, per visualizzare gli errori di classificazione tra le diverse classi.

Si è mantenuta come principale metrica di valutazione l'**F1-Score Macro**, che ha raggiunto nel primo stadio un valore di **0.75** e nel secondo stadio un valore di **0.32**. Questi risultati (tabella 3.1) indicano una buona capacità del modello di distinguere tra le cellule sane e quelle infette, tuttavia le performance del secondo stadio sono inferiori a causa del grande numero di **falsi negativi** in output al primo stadio (come visto in figura 3.7) e per la complessità intrinseca della classificazione tra le sottoclassi infette.

Stage	Class	Precision	Recall	F1	Support
Stage 1	0	0.99	0.96	0.98	14,195
	1	0.40	0.66	0.50	507
Stage 2	0	0.74	0.53	0.62	193
	1	0.19	0.22	0.21	18
	2	0.31	0.53	0.39	66
	3	0.24	0.36	0.29	25
	4	0.13	0.10	0.11	31
Accuracy				0.95/0.46	
Macro Avg				0.75/0.32	
Weighted Avg				0.96/0.48	

Tabella 3.1: Tabella di valutazione delle performance del modello. Le metriche sono calcolate per ciascuna classe nel primo e nel secondo stadio.

Tutte le metriche e i risultati sono stati salvati automaticamente in un file di log, per garantire la tracciabilità e la riproducibilità degli esperimenti.

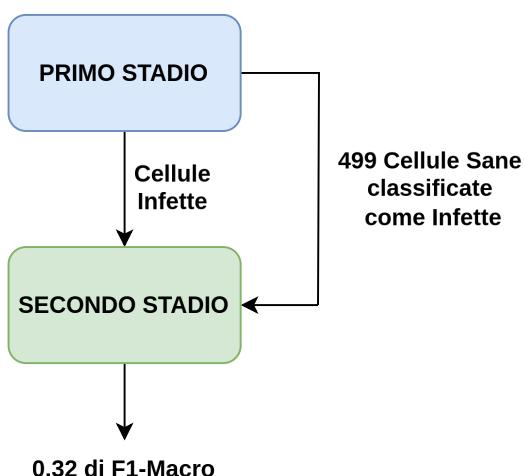


Figura 3.7: Grafico esplicativo del sistema a cascata.

Le matrici di confusione riportate in Figura 3.8 e Figura 3.9 illustrano i risultati ottenuti rispettivamente dal primo e dal secondo classificatore del sistema a cascata.

Nel primo classificatore (Figura 3.8), che ha il compito di distinguere le cellule sane (**globuli rossi** e **globuli bianchi**) da **quelle infette**, si osserva una netta separazione tra le due classi. La maggioranza delle cellule sane viene correttamente classificata (**13.696** su **14.195**), mentre le **cellule infette**, pur essendo in netta minoranza, vengono identificate con un'accuratezza soddisfacente (**337** su **507**), pur con **170 falsi positivi**. Questo risultato è comunque positivo, considerando lo sbilanciamento del dataset, e dimostra l'efficacia del primo stadio nel filtrare correttamente la maggior parte delle cellule sane.

La matrice di confusione del secondo classificatore (Figura 3.9), che si occupa della classificazione fine delle **cellule infette**, mostra invece una maggiore difficoltà nel distinguere tra le varie sottoclassi (**trophozoite**, **ring**, **difficult**, **schizont** e **gametocyte**). Alcune classi, in particolare i **ring** e i **trophozoite**, vengono spesso confuse tra loro, come evidenziato dagli alti valori fuori diagonale. Questo comportamento è atteso, data la somiglianza morfologica tra queste tipologie cellulari.

Complessivamente, l'approccio a cascata ha permesso di isolare il problema della classificazione **fine delle cellule infette**, migliorando la robustezza del sistema rispetto a una classificazione a 7 classi unica. I risultati ottenuti nel primo stadio sono solidi, mentre nel secondo stadio le performance sono minori a causa della complessità intrinseca della classificazione tra le sottoclassi infette.

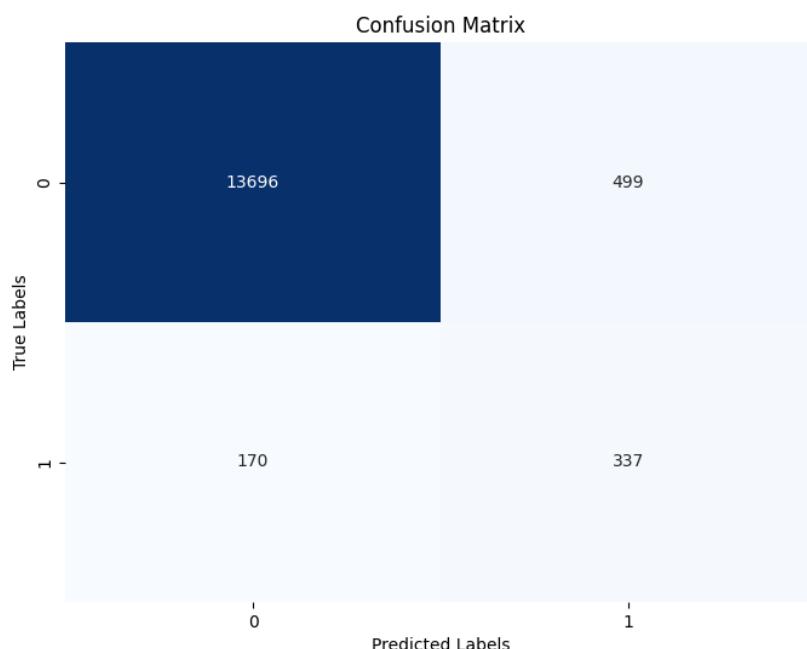


Figura 3.8: Matrice di Confusione del primo Classificatore.

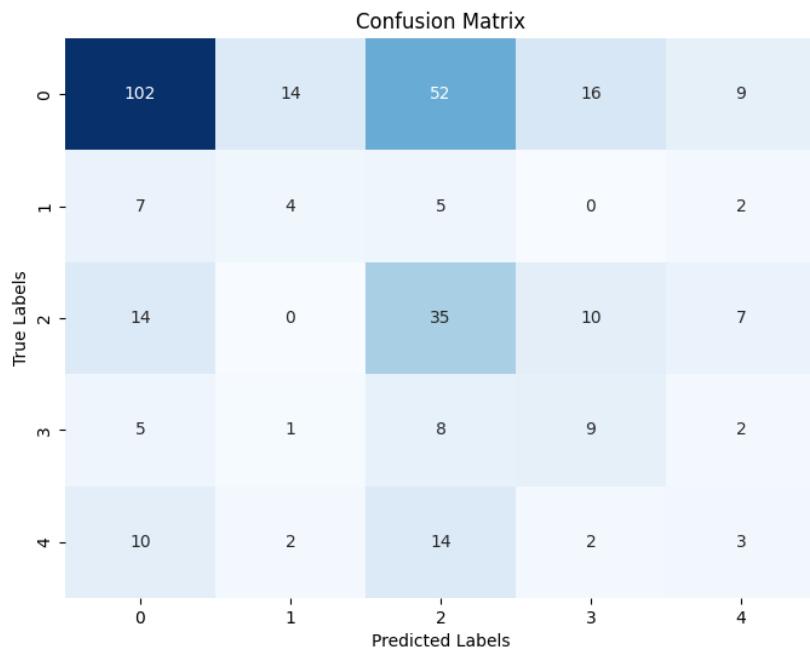


Figura 3.9: Matrice di Confusione del secondo Classificatore.