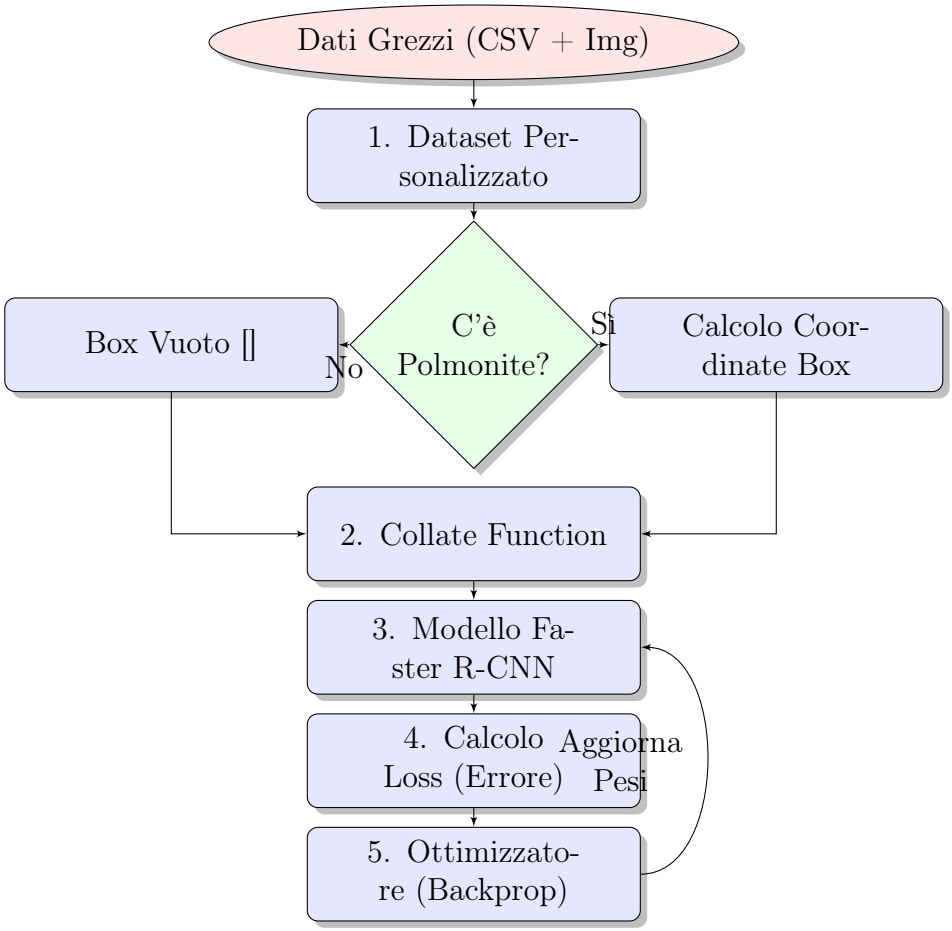
1 Analisi Dettagliata dell'Implementazione

Il cuore del progetto risiede nello script `train_detection.py`. Questo codice gestisce l'intero ciclo di vita dell'addestramento, dalla preparazione dei dati grezzi fino al salvataggio del modello neurale. Di seguito viene analizzato il flusso logico e le singole componenti software.

1.1 Schema del Processo di Training

Il seguente diagramma illustra il flusso dei dati all'interno della pipeline di addestramento implementata:



1.2 Spiegazione dei Componenti Software

1.2.1 1. Configurazione e Setup

In questa fase iniziale vengono definite le "regole" dell'addestramento.

```
BATCH_SIZE = 8
DEVICE = torch.device('cuda') if torch.cuda.is_available() else torch.device
    ('cpu')
```

- **Batch Size (8):** A differenza della classificazione classica, l'Object Detection richiede molta memoria VRAM per memorizzare le coordinate di multiple box per ogni immagine. Un valore di 8 è un compromesso ottimale per la GPU Tesla P100.

- **Device:** Il codice verifica automaticamente la presenza di accelerazione hardware (CUDA) per garantire tempi di esecuzione ridotti.

1.2.2 2. Il Dataset Personalizzato

La classe `PneumoniaDetectionDataset` trasforma i dati grezzi in tensori comprensibili dal modello. Il problema principale risolto da questa classe è la natura "disordinata" del CSV, dove un singolo paziente può avere più righe (anomalie multiple).

Passaggi chiave:

1. **Raggruppamento:** Il codice identifica i `patientId` univoci.
2. **Estrazione:** Per ogni immagine, estrae tutte le righe corrispondenti nel CSV.
3. **Conversione Geometrica:** Converte le coordinate dal formato sorgente (x, y, w, h) al formato richiesto da PyTorch $(x_{min}, y_{min}, x_{max}, y_{max})$.
4. **Gestione Casi Negativi:** Se il paziente è sano, restituisce una lista vuota per i Bounding Box. Questo insegna al modello a *non* disegnare nulla in assenza di patologia.

1.2.3 3. La Funzione `collate_fn`

```
def collate_fn(batch):
    return tuple(zip(*batch))
```

Questa funzione è cruciale per prevenire errori di runtime. Normalmente, i DataLoader cercano di "impilare" (stack) i dati in un unico tensore. Tuttavia, nell'Object Detection, le immagini hanno un numero variabile di annotazioni (es. Img A ha 2 box, Img B ne ha 0). Non essendo uniformi, non possono essere impilate. La `collate_fn` istruisce PyTorch a mantenere i dati come una lista di tuple (immagini e target separati), preservando la struttura variabile.

1.2.4 4. Il Modello: Faster R-CNN

L'architettura utilizzata è basata sul principio del **Transfer Learning**.

```
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True
    )
# Sostituzione della "testa" del modello
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes
    =2)
```

Viene caricato un modello pre-addestrato su COCO (oggetti comuni). La parte finale della rete (il classificatore) viene rimossa e sostituita con un nuovo layer dimensionato per 2 sole classi: *Sfondo* e *Polmonite*. Questo permette al modello di sfruttare la capacità di estrazione delle feature (linee, forme) appresa in precedenza, applicandola al nuovo contesto medico.

1.2.5 5. Il Ciclo di Addestramento (Training Loop)

È il processo iterativo in cui avviene l'apprendimento vero e proprio.

- **Forward Pass:** Il modello riceve un batch di immagini e predice i bounding box.
- **Loss Calculation:** Poiché Faster R-CNN è un modello complesso, calcola automaticamente un dizionario di errori (Losses), che include:
  - Errore di classificazione (è polmonite?).

  - Errore di regressione (il rettangolo è nella posizione giusta?).

- **Backward Pass & Optimizer:** Gli errori vengono sommati e utilizzati per calcolare il gradiente (direzione di correzione). L'ottimizzatore SGD (Stochastic Gradient Descent) aggiorna i pesi della rete per minimizzare l'errore nel passaggio successivo.