XRayAnalyzer: Agente Radiologo Intelligente

Documentazione Tecnica e Implementazione

Aurora Guida

29 novembre 2025

Indice

Capitolo 1

Introduzione e Obiettivi

1.1 Visione del Progetto

Il progetto **XRayAnalyzer** nasce per superare i limiti dei tradizionali sistemi di classificazione di immagini mediche. Mentre una classica CNN fornisce solo un output binario ("Sano/Malato"), il nostro sistema è progettato come un **Agente Intelligente** in grado di:

1. **Vedere:** Localizzare con precisione le anomalie (bounding box) usando un modello Faster R-CNN addestrato su HPC.
2. **Consultare:** Accedere a una Knowledge Base medica (RAG - Retrieval Augmented Generation) per contestualizzare la diagnosi.
3. **Spiegare:** Fornire un referto testuale completo, combinando evidenze visive e protocolli clinici.

Capitolo 2

Dettagli Implementativi dei File Sorgente

In questo capitolo analizziamo il codice sorgente sviluppato, spiegando la logica dietro ogni modulo principale.

2.1 1. Addestramento: `train_detection.py`

Questo script è responsabile della creazione del modello di visione. Viene eseguito sull'infrastruttura HPC per sfruttare l'accelerazione GPU.

Funzionalità Chiave:

- Gestione del Dataset RSNA (parsing del CSV e raggruppamento box per paziente).
- Definizione del modello Faster R-CNN con backbone ResNet50.
- Ciclo di training con salvataggio checkpoint per ogni epoca.

```
# Configurazione Iperparametri
BATCH_SIZE = 8
NUM_EPOCHS = 5
DEVICE = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

class PneumoniaDetectionDataset(Dataset):
    def __getitem__(self, idx):
        # Logica per gestire la relazione "Uno-a-Molti"
        img_id = self.image_ids[idx]
        records = self.csv[self.csv['patientId'] == img_id]

        boxes = []
        for i, row in records.iterrows():
            if row['Target'] == 1:
                # Conversione coordinate: da (x,y,w,h) a (xmin,ymin,xmax,ymax)
                x_min = row['x']
                y_min = row['y']
                x_max = row['x'] + row['width']
                y_max = row['y'] + row['height']
                boxes.append([x_min, y_min, x_max, y_max])

        # Gestione Pazienti Sani (Lista box vuota)
        if len(boxes) == 0:
            boxes = torch.zeros((0, 4), dtype=torch.float32)

        return img, target
```

Listing 2.1: Estratto da train_detection.py: Configurazione e Dataset

2.2 2. Backend Server: `app.py`

Il file `app.py` è il punto di ingresso dell'applicazione Web basata su Flask. Agisce da orchestratore tra l'utente, il modello di visione e l'agente logico.

Funzionalità Chiave:

- Caricamento del modello `.pth` all'avvio.
- Gestione della rotta `/predict` per ricevere le immagini.
- Coordinamento: chiama prima la Visione, poi passa i dati all'Agente per il report.

```python
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # 1. Analisi Visiva (Vision Tool)
        # Utilizza il modello caricato per trovare box e confidenza
        boxes, scores = get_prediction(model, image_bytes, DEVICE, threshold=0.5)

        # Calcolo statistiche per l'Agente
        num_boxes = len(boxes)
        max_score = max(scores) if scores else 0.0

        # 2. Generazione Report (Reasoning Tool)
        # L'agente usa i dati visivi per interrogare la Knowledge Base
        report_data = agent.generate_report(num_boxes, max_score)

        # 3. Risposta JSON al Frontend
        return jsonify({
            'boxes': boxes,
            'scores': scores,
            'report': report_data
        })
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Listing 2.2: Estratto da web_app/app.py

2.3 3. Logica dell'Agente: `agent.py`

Questo file implementa il "cervello" del sistema. Utilizza un approccio RAG (Retrieval Augmented Generation) semplificato e deterministico per garantire affidabilità in ambito medico.

Funzionalità Chiave:

- Caricamento della Knowledge Base da file di testo (`info_mediche.txt`).
- Logica decisionale: Se rileva anomalie → Cerca protocolli polmonite. Se sano → Restituisce referto negativo standard.

```python
class MedicalAgent:
    def generate_report(self, num_boxes, max_score):
        # CASO 1: PAZIENTE SANO
        if num_boxes == 0:
            return {
                "titolo": "REFERTO RADIOLOGICO: NEGATIVO",
                "colore": "#10b981", # Verde
                "testo": self.knowledge.get("SANO")
            }

        # CASO 2: PATOLOGIA RILEVATA
        # Recupero info cliniche (RAG Simulato)
```

```
        info_cliniche = self.knowledge.get("POLMONITE_BATTERICA")

        testo_report = f"""
        RILEVAMENTI VISIVI:
            Identificate {num_boxes} aree di consolidamento.
            Confidenza modello: {max_score:.1%}.

        NOTE CLINICHE E PROTOCOLLO:
        {info_cliniche}
        """

        return {
            "titolo": "RILEVATA POSSIBILE POLMONITE",
            "colore": "#ef4444", # Rosso
            "testo": testo_report
        }
```

Listing 2.3: Estratto da web_app/agent.py

2.4 4. Utility Modello: `model_utils.py`

Per mantenere il codice pulito, le funzioni di caricamento e inferenza pura sono state isolate in questo modulo.

Funzionalità Chiave:

- `get_model()`: Ricostruisce l'architettura Faster R-CNN da zero (necessario perché salviamo solo i pesi *state_ dict*).
- `get_prediction()`: Gestisce il pre-processing dell'immagine (trasformazione in tensore) e il post-processing (filtro dei risultati per soglia di confidenza).

Capitolo 3

Integrazione e Deployment

3.1 Architettura Docker

Il sistema è containerizzato per garantire la riproducibilità. Il `Dockerfile` definisce l'ambiente Linux, installa le dipendenze di sistema (librerie GL per OpenCV) e le librerie Python elencate in `requirements.txt`.

- **Base Image:** Python 3.9 Slim / Ubuntu.
- **Dipendenze:** Flask, Torch, Torchvision, Pillow.
- **Porte:** Espone la porta 5000 per l'interfaccia web.

3.2 Interfaccia Utente (Frontend)

L'interfaccia (HTML/JS) è progettata per essere responsiva e chiara. Il file `script.js` gestisce la comunicazione asincrona con il backend:

1. Invia l'immagine via POST.
2. Riceve il JSON con coordinate e report.
3. Disegna i rettangoli sul `<canvas>` sopra l'immagine.
4. Inietta il testo del report nel box dedicato, cambiando dinamicamente il colore (Rosso/Verde) in base alla diagnosi.

Capitolo 4

Conclusioni

L'architettura modulare presentata permette di separare nettamente le responsabilità: `train_detection.py` si occupa dell'apprendimento profondo su HPC, mentre `app.py` e `agent.py` gestiscono la logica applicativa e l'interazione utente. L'uso di Docker unifica il tutto in un pacchetto facilmente distribuibile.