AI-Powered Anomaly Detection in Medical Imagery

WIDS: Project Report

Prince Chouhan (22B3970)
IIT Bombay

Department of Electrical Engineering

https://github.com/healthcare-ai-vision/Prince/tree/main

Contents

Introduction

1.1 Background and Motivation

Medical imaging has revolutionized healthcare by allowing non-invasive visualization of internal anatomy. However, the exponential growth in the volume of medical data—ranging from X-rays and MRIs to histopathological slides—has outpaced the capacity of human experts. Manual interpretation is time-consuming, prone to inter-observer variability, and susceptible to fatigue-induced errors.

Computer Aided Diagnosis (CAD) systems serve as a "second pair of eyes" for radiologists and pathologists. By leveraging Artificial Intelligence (AI), specifically Convolutional Neural Networks (CNNs), these systems can detect subtle patterns and features that may be imperceptible to the human eye.

1.2 Project Objectives

The primary goal of this project was to build a vertically integrated AI pipeline. The specific sub-objectives were:

1. **Image Quality Assessment**: To understand how to manipulate raw pixel data to improve diagnostic visibility.
2. **Automated Triage**: To develop a high-speed classifier for skin lesions to prioritize malignant cases.
3. **Quantitative Pathology**: To automate the detection of immune cells in tissue samples, a task that is tedious for humans but critical for cancer staging.

1.3 Report Structure

This report is organized chronologically, reflecting the learning curve from basic signal processing to advanced neural network training:

- **Chapter 2** outlines the technical environment and software stack.
- **Chapter 3** details fundamental image processing algorithms (histograms, filtering).
- **Chapter 4** presents the deep learning methodology for skin cancer classification.
- **Chapter 5** discusses the object detection framework for histopathology.
- **Chapter 6** summarizes the findings and proposes future research directions.

Technical Stack & Methodology

2.1 Development Environment

Given the computational intensity of training Deep Neural Networks (DNNs), this project utilized a hybrid environment.

2.1.1 Hardware Acceleration

Deep learning models rely heavily on matrix multiplications. We utilized **Google Colab Pro**, utilizing an **NVIDIA Tesla T4 GPU** with 16GB of VRAM and 2560 CUDA cores. This hardware acceleration reduced training times from hours (on CPU) to minutes.

2.1.2 Software Frameworks

- **Python 3.10**: Selected for its rich ecosystem of data science libraries.
- **OpenCV (Open Source Computer Vision Library)**: Used for efficient image I/O, resizing, and implementing classical filtering kernels.
- **PyTorch**: The backend tensor computation engine. It provides dynamic computation graphs, making it ideal for research and prototyping.
- **Ultralytics YOLOv8**: A cutting-edge vision model. While traditionally an object detector, YOLOv8 introduced a "Classification" head (YOLOv8-cls), which was chosen for its balance of speed (FLOPs) and accuracy compared to heavier models like ResNet50.

2.2 Data Management

Medical data requires strict organization. We used **Roboflow** for:

1. **Versioning**: Keeping track of different dataset splits.
2. **Preprocessing**: Auto-orienting and resizing images before download.
3. **Format Conversion**: Converting annotations between JSON, XML, and YOLO (.txt) formats.

Week 2: Medical Image Processing

3.1 Theoretical Foundation

Before a neural network can analyze an image, the image often requires restoration. Medical images are discrete representations of continuous physical signals, often corrupted by noise during acquisition.

3.2 Histogram Analysis and Equalization

An image histogram is a discrete function $h(r_k) = n_k$, where r_k is the k-th gray level and n_k is the number of pixels with that gray level.

In medical imaging, contrast is often poor. For example, in an X-ray, the difference between soft tissue and a tumor might be subtle. We utilized histogram analysis to identify these distributions.

3.2.1 Code Implementation

The following Python code demonstrates extracting grayscale intensity distributions using OpenCV:

```python
import cv2
import matplotlib.pyplot as plt

# Load image in grayscale mode
img = cv2.imread("medical_sample.png", 0)

# Calculate histogram
# parameters: [image], [channel], [mask], [histSize], [ranges]
hist = cv2.calcHist([img], [0], None, [256], [0, 256])

plt.plot(hist)
plt.title("Grayscale Intensity Distribution")
plt.xlabel("Pixel Value (0-255)")
plt.ylabel("Frequency")
plt.show()
```

Listing 3.1: Histogram Calculation

3.3 Noise Simulation and Restoration

To test the robustness of our processing pipeline, we artificially degraded high-quality images with noise models mathematically similar to real-world sensor errors.

3.3.1 Gaussian Noise

Gaussian noise arises from electronic circuit noise and sensor heat. It is additive noise following a normal distribution $p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\mu)^2/2\sigma^2}$.

- **Restoration Technique**: Gaussian Blur. This is a low-pass filter that convolves the image with a Gaussian kernel. It effectively smooths out high-frequency noise but blurs edges.

3.3.2 Salt and Pepper Noise

Impulse noise, often caused by analog-to-digital converter errors, manifests as random white and black pixels.

- **Restoration Technique**: Median Filter. Unlike Gaussian blur, the median filter is non-linear. It replaces a pixel with the median value of its neighbors.
- **Advantage**: It is excellent at preserving edges while removing impulse noise.

```python
import numpy as np

# Apply Gaussian Blur (Kernel size 5x5)
gaussian_blur = cv2.GaussianBlur(noisy_image, (5, 5), 0)

# Apply Median Filter (Kernel size 5)
median_filtered = cv2.medianBlur(noisy_image, 5)

# Display comparisons
cv2.imshow("Restored", np.hstack([gaussian_blur, median_filtered]))
```

Listing 3.2: Noise Filtering Implementation

Week 3: Deep Learning for Skin Cancer Classification

4.1 Problem Definition

Melanoma is the deadliest form of skin cancer, but it has a high survival rate if treated early. The visual distinction between a benign mole and a malignant melanoma is often based on the "ABCDE" rule (Asymmetry, Border, Color, Diameter, Evolving). Our goal was to train a CNN to learn these features automatically.

4.2 Dataset: ISIC Archive (Subset)

We utilized the `skin-ga5ww` dataset hosted on Roboflow, which is a curated subset of the ISIC (International Skin Imaging Collaboration) archive.

Split	**Image Count**	**Percentage**
Training Set	10,088	82%
Validation Set	1,442	12%
Testing Set	720	6%

Table 4.1: Dataset Distribution

The classes were balanced to prevent model bias: 1. **Benign (Basal Cell Carcinoma)** 2. **Malignant (Melanoma)**

4.3 YOLOv8 Classification Architecture

While YOLO (You Only Look Once) is famous for object detection, the v8 iteration introduced a streamlined classification head.

- **Backbone**: Modified CSPDarknet53. It uses Cross-Stage Partial connections to reduce computation while maintaining rich gradient flow.
- **Stem**: A convolutional layer with stride 2 to reduce spatial dimensions.
- **Head**: Instead of bounding box regressors, the classification head uses Global Average Pooling followed by a Linear Layer and Softmax activation.

4.4 Training Protocol

We used Transfer Learning, initializing the model with weights pretrained on the ImageNet dataset. This allows the model to leverage previously learned features (edges, textures) and converge faster on the medical task.

- **Model**: `yolov8n-cls.pt` (Nano variant, approx 3.2M parameters).
- **Optimizer**: AdamW (Adam with Weight Decay) to prevent overfitting.
- **Learning Rate**: Initial $lr = 0.001$.
- **Image Size**: Resized to 224×224.
- **Epochs**: 10.

4.5 Experimental Results

The training process was monitored via the loss function (Cross-Entropy Loss).

4.5.1 Loss Convergence

The training loss dropped sharply in the first 3 epochs, stabilizing around Epoch 8.

- **Initial Loss**: 0.2534
- **Final Loss**: 0.0728

4.5.2 Accuracy Metrics

On the validation set, the model achieved:

- **Top-1 Accuracy**: 97.9%. This means in 97.9% of cases, the model's highest probability prediction was the correct class.
- **Inference Speed**: The model processed images at ≈ 2.5 ms per image, translating to roughly 400 FPS, making it suitable for real-time video analysis in clinical settings.

Week 4: Object Detection in Histopathology

5.1 The Challenge of WSI

Whole Slide Images (WSI) are high-resolution scans of tissue slides. Unlike the classification task where we categorize the *whole* image, here we must find *specific objects* (cells) within the image.

5.2 Target Objects

We focused on detecting two types of immune cells:

1. **Lymphocytes**: Small, round inflammatory cells.
2. **Plasma Cells**: Larger cells with an eccentric nucleus.

The presence and density of these cells are key indicators of the Tumor-Infiltrating Lymphocyte (TIL) score, which predicts patient survival in many cancers.

5.3 Methodology: YOLOv8 Detection

For this task, we used the standard YOLOv8 detection model. The model predicts:

$$P(class|object) \cdot IoU_{pred}^{truth}$$

It outputs a vector $(x, y, w, h, confidence, class)$ for every detected object.

5.3.1 Dataset Annotation

The dataset `wsiroisimages` was annotated using bounding boxes. The labels were stored in YOLO format text files, where coordinates are normalized to $[0, 1]$:

```
<class_id> <x_center> <y_center> <width> <height>
0 0.716 0.322 0.054 0.062
```

5.3.2 Verification Script

Before training, it is crucial to verify that labels align correctly with images. We implemented a visualization script in Python.

```python
def visualize_labels(image_path, label_path):
    img = cv2.imread(image_path)
    h, w, _ = img.shape

    with open(label_path, 'r') as f:
        lines = f.readlines()

    for line in lines:
        parts = line.strip().split()
        class_id = int(parts[0])
        # Denormalize coordinates
        x_center, y_center = float(parts[1]) * w, float(parts[2]) * h
        width, height = float(parts[3]) * w, float(parts[4]) * h

        # Calculate top-left corner
        x1 = int(x_center - width/2)
        y1 = int(y_center - height/2)
        x2 = int(x_center + width/2)
        y2 = int(y_center + height/2)

        # Draw rectangle
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)

    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.show()
```

Listing 5.1: Ground Truth Visualization

5.4 Analysis of Results

The detection model faced challenges unique to microscopy:

- **Occlusion**: Cells often overlap.
- **Similarity**: Lymphocytes look very similar to other nuclei.

Despite these challenges, the YOLOv8 model successfully identified dense clusters of cells. The "Grid-based" approach of YOLO is particularly well suited here compared to Region Proposal Networks (like Faster R-CNN) because it handles high object density more efficiently.

Conclusion and Future Scope

6.1 Summary of Achievements

This project successfully demonstrated the viability of AI in the medical domain.

1. **Preprocessing**: We established that simple spatial filtering can significantly recover image quality, a crucial step before AI processing.
2. **Classification**: We trained a model that achieved **97.9% accuracy** on skin cancer detection. This implies that for every 100 cases, the model correctly diagnosed approx 98, a performance comparable to junior dermatologists.
3. **Detection**: We successfully implemented a pipeline to count immune cells, automating a task that usually takes pathologists hours.

6.2 Future Scope

6.2.1 Model Improvements

- **Data Augmentation**: Implementing "Test Time Augmentation" (TTA) to average predictions over rotated versions of the input image would likely increase robustness.
- **Explainability (XAI)**: Implementing Grad-CAM (Gradient-weighted Class Activation Mapping) to visualize *where* the model is looking. Doctors need to know if the model is looking at the lesion or the ruler in the image.

6.2.2 Deployment

The next logical step is to containerize the model using **Docker** and deploy it as a REST API using **FastAPI**. This would allow mobile applications to send images to our server and receive diagnostic predictions in real-time.