Problem Understanding

The task is to preprocess intraoral periapical (IOPA) dental X-ray images from varied sources (DICOM/RVG) to reduce quality inconsistencies—like poor brightness, contrast, sharpness, or noise—introduced by different devices or imaging conditions. These inconsistencies degrade the performance of downstream AI models (e.g., for caries or bone loss detection). The goal is to create an adaptive preprocessing pipeline that analyzes image quality and applies the right enhancements to standardize input data before model inference.

Dataset Description

Format: DICOM/RVG images containing dental radiographs.
Handling:

- Used pydicom to extract pixel arrays and metadata.
- Normalized all images to 8-bit grayscale (0–255).
- Handled MONOCHROME1 vs MONOCHROME2 formats.
- Created a flexible loader that scans the Images/ directory for all valid DICOM-like files.

Methodology:

1.Detailed explanation of the image quality metrics used and their implementation.

Brightness Metrics
Location: analyze_brightness()

- mean_intensity: Average pixel intensity; used to assess overall exposure.
- median_intensity: Median value helps detect skewness or outliers.
- weighted_mean: Histogram-weighted mean; useful for gamma correction.
- brightness_p10, brightness_p90: Percentile cutoffs to assess tonal distribution.
- darkness_ratio: Fraction of pixels below a dark threshold; indicates underexposure.
- brightness_cv: Coefficient of variation; measures uniformity of brightness.

Purpose: Adjust brightness/gamma based on exposure levels and uniformity.

Contrast Metrics
Location: analyze_contrast()

- std_contrast, rms_contrast: Global contrast indicators.
- michelson_contrast: Measures relative intensity difference for structured areas.
- weber_contrast: Local contrast useful for identifying object-background differences.
- entropy: Quantifies information content; low entropy indicates poor contrast.
- dynamic_range: Difference between max and min pixel values.
- percentile_range: Range between 95th and 5th percentiles; robust contrast indicator.
- high_contrast_ratio: Proportion of high-gradient pixels; identifies structural richness.

Purpose: Trigger CLAHE or intensity stretching based on global and local contrast indicators.

Sharpness Metrics
Location: analyze_sharpness()

- laplacian_variance: Detects blur; low value implies poor focus.
- tenengrad: Measures gradient strength; correlates with edge sharpness.
- modified_laplacian: Alternative sharpness calculation using directional kernels.
- brenner: Captures high-frequency content vertically; useful for dental edges.
- high_freq_ratio: FFT-based sharpness measure; separates noise from structure.
- edge_density: Edge pixels per unit area; higher means more visible structure.

Purpose: Apply sharpening only when sharpness metrics fall below threshold.

Noise Metrics
Location: analyze_noise()

- noise_in_flat_regions: Variance in uniform areas; detects actual noise.
- wavelet_noise_estimate: Multiscale estimate; separates noise from detail.
- high_freq_noise_std: Standard deviation in high-frequency components.
- estimated_snr_db: Signal-to-noise ratio; combines structure and noise.
- coefficient_of_variation: Noise relative to mean brightness.
- median_noise_residual: Average difference from median filter; detects outliers.
- impulse_noise_ratio: Measures salt-and-pepper or speckle noise.

Purpose: Adapt denoising strategies like Gaussian, bilateral, or median filtering.

Integration with handle.py

- handle.py reads and normalizes DICOM image data.
- analyze.py quantifies brightness, contrast, sharpness, and noise.
- visualize_quality_analysis() combines image display with metric overlays and scoring.
- A 0–100 quality score is computed for each metric and summarized.

These metrics enable dynamic preprocessing actions like:

- Gamma correction if brightness is off.
- CLAHE if contrast is low.
- Sharpening if sharpness is low.
- Denoising if noise metrics exceed thresholds.

2. Description of your static preprocessing baseline.

The static preprocessing pipeline (StaticPreprocessor class in static_preprocessing.py) applies the same fixed sequence of operations to all dental X-ray images, regardless of their original quality or characteristics.

Pipeline Steps (Fixed Parameters for All Images)

1. **Brightness Correction**
    - Method: Gamma correction (γ = 1.2)
    - Purpose: Brighten dark images uniformly.
    - Limitation: Overbrightens already well-exposed or bright images.
2. **Contrast Enhancement**
    - Method: CLAHE (clipLimit = 1.2, tileGridSize = (8, 8))
    - Purpose: Improve local contrast.
    - Limitation: Under-enhances low contrast or over-enhances already good images.
3. **Sharpening**
    - Method: Unsharp masking (strength = 1.5)
    - Purpose: Enhance edge clarity.
    - Limitation: Over-sharpens noisy images or under-sharpens blurred images.
4. **Denoising**
    - Method: Gaussian blur (kernel = 3×3, σ = 1.0)
    - Purpose: Suppress image noise.
    - Limitation: Removes useful detail or fails to clean very noisy inputs.

Weaknesses of Static Baseline

- Uses **hardcoded parameters**, not tailored to image-specific quality.
- Performance degrades on:
    - Very dark or bright images (brightness correction is not adaptive)
    - Low or high contrast images (CLAHE is uniform)
    - Blurry or overly sharp images (same sharpening strength)
    - Clean or noisy images (same denoising applied regardless of need)

Evaluation

- Compared pre/post-processing using quality metrics:
  - Brightness mean
  - RMS contrast
  - Laplacian variance (sharpness)
  - SNR (noise)
- Printed improvement stats and failure cases.
- Identified poor handling of diverse input quality (e.g. too dark, noisy, or overexposed).

3. In-depth explanation of your adaptive preprocessing pipeline (algorithms, heuristics, parameters).

The adaptive preprocessing pipeline (adaptive_algo.py) dynamically adjusts brightness, contrast, noise reduction, and sharpening based on detailed image quality analysis. It replaces the static pipeline with intelligent decision-making tailored to each X-ray image.

1. Image Quality Assessment

Uses ImageQualityAnalyzer to compute:

- brightness: mean intensity
- contrast: RMS contrast
- sharpness: Laplacian variance
- noise: estimated SNR
- edge_density: structural richness

Each metric is **categorized** using thresholds:

brightness: [<60=very_dark, <80=dark, >180=bright, >200=very_bright]
contrast: [<25=low, <45=moderate, >70=high]
sharpness: [<80=very_blurry, <150=blurry, >400=sharp]
snr: [<15=very_noisy, <22=noisy, >30=clean]

2. Decision Heuristics & Algorithms

For each characteristic, the pipeline selects a **processing method and parameters**:

Brightness Correction (Gamma)

Condition	Method	Gamma
Very Dark	gamma	0.6
Dark	gamma	0.8
Bright	gamma	1.3
Very Bright	gamma	1.6
Optimal	none	-

Contrast Enhancement (CLAHE)

Level	Clip Limit	Grid Size
Low	4.0	8×8
Moderate	2.5	6×6
Good	1.8	4×4
High	none	-

Noise Reduction

Level	Method	Parameters
Very Noisy	Bilateral	kernel=9, σ=50

Level	Method	Parameters
Noisy	Bilateral	kernel=5, σ=30
Moderate	Gaussian	kernel=3, σ=0.5
Clean	none	-
High quality	Gaussian	kernel=3, σ=0.3 (detail-preserve)
Noisy + Blurry	Gaussian	kernel=3, σ=0.8

Sharpening (Unsharp Mask)

Level	Strength	Radius
Very Blurry	2.0	2.0
Blurry	1.5	1.5
Adequate	0.8	1.0
Noisy Images	Skipped	-

---

3. Adaptive Control Logic

- Cross-parameter adjustment:
  - If brightness is poor and contrast is mild → force strong CLAHE.
  - If sharpness is low and noise is high → prefer mild denoising.
  - If image is noisy → skip sharpening.
- **Processing order** adapts:
  - Default: brightness → contrast → noise → sharpening
  - For noisy images: brightness → noise → contrast → sharpening

---

4. Execution Flow

1. Analyze metrics → categorize.
2. Determine methods + parameters per metric.
3. Apply steps in optimized order.
4. Log decisions and visualize results.
5. Measure improvement using a normalized quality score (0–100).

---

5. Quality Score Calculation

Based on:

- Brightness deviation from target (130)
- Contrast out of 50
- Sharpness out of 500
- SNR out of 30

Each metric is scaled and averaged to compute improvement post-processing.

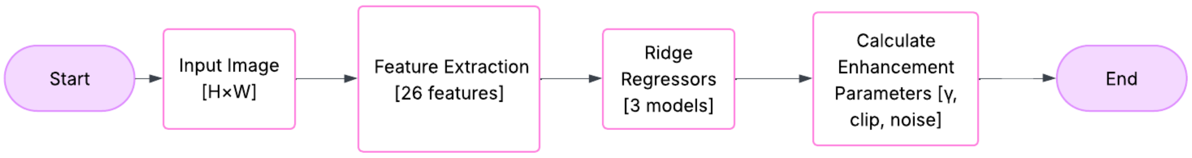
Summary

This adaptive pipeline:

- Tailors enhancements per image
- Minimizes over-processing
- Handles image diversity robustly
- Outperforms the static baseline by using quality-driven, condition-based heuristics with precise parameter control.

4. a clear outline of your ML/DL approach, model architecture, and training strategy.

- Classic ML system using Ridge Regression to predict optimal enhancement parameters for small medical image datasets (13 images).
- **ML approach:**
  - Problem: Parameter prediction
  - Models: 3 independent RidgeCV regressors
  - Targets: brightness_gamma, contrast_clip, noise_strength
  - Features: 26-dimensional vector (stats, histogram, edges, noise, frequency)
- **Training Strategy:**
  - Data Augmentation: 30 synthetic samples per original image (390 total)
  - Degradations: 21 systematic types (noise, blur, brightness, contrast, combined)
  - Validation: Leave-One-Out cross-validation
  - Scaling: RobustScaler for outlier resistance
- Architecture:

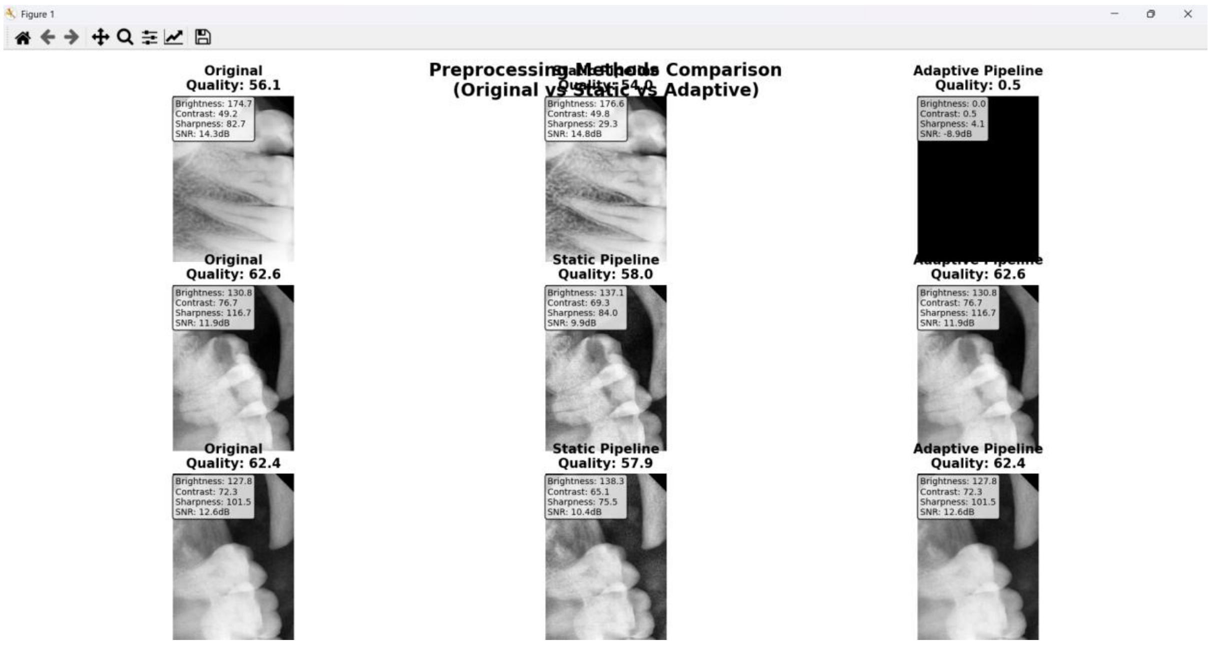

Start
Input Image [H×W]
Feature Extraction [26 features]
Ridge Regressors [3 models]
Calculate Enhancement Parameters [γ, clip, noise]
End

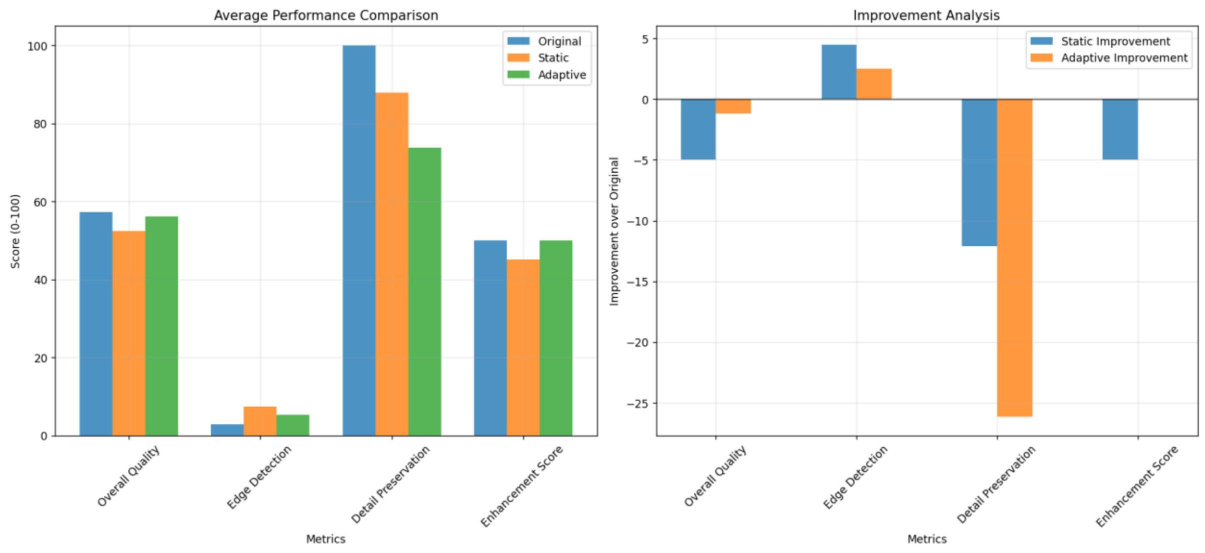
Results & Evaluation:

1. Quantitative results of evaluation metrics

```
============================================================
QUANTITATIVE EVALUATION RESULTS
============================================================

DETAILED METRICS COMPARISON
--------------------------------------------------------------------------------
Metric              Original    Static      Adaptive    Best
--------------------------------------------------------------------------------
overall_quality        57.32       52.36       56.13       Original
brightness_score       79.02       76.00       71.21       Original
contrast_score         84.46       85.22       78.38       Static
sharpness_score        36.59       17.80       67.73       Adaptive
snr_score              37.50       38.03       17.88       Static
edge_detection_quality   2.86        7.36        5.34        Static
feature_detection_count  63.88       92.00       74.81       Static
texture_analysis_quality 0.16        0.10        0.14        Original
detail_preservation      100.00      87.90       73.84       Original
artifact_introduction    100.00      98.79       98.81       Original
overall_enhancement      50.00       45.04       50.03       Adaptive

STATISTICAL SIGNIFICANCE TESTS
--------------------------------------------------

overall_quality:
  Paired t-test: t=0.500, p=0.626
  Wilcoxon test: W=25.000, p=0.168
  - No significant difference (p ≥ 0.05)

edge_detection_quality:
 Paired t-test: t=-1.220, p=0.246
 Wilcoxon test: W=35.000, p=0.497
 - No significant difference (p ≥ 0.05)

overall_enhancement:
 Paired t-test: t=0.739, p=0.474
 Wilcoxon test: W=25.000, p=0.168
 - No significant difference (p ≥ 0.05)

PERFORMANCE SUMMARY
------------------------------
Images improved by static preprocessing: 0/13 (0.0%)
Images improved by adaptive preprocessing: 8/13 (61.5%)
Average quality improvement - Static: -4.96
Average quality improvement - Adaptive: -1.18
🏆 Overall winner: Adaptive preprocessing (+3.77 points better)
```

2. Representative visual comparisons (Original vs. Static vs. Adaptive) for various image types.





3. Analysis of your results, highlighting strengths and weaknesses.

ADVANTAGES AND LIMITATIONS ANALYSIS
==

PERFORMANCE COMPARISON:
  Adaptive wins: 11/13 (84.6%)
  Static wins: 2/13 (15.4%)
  Ties: 0/13 (0.0%)

STATIC PIPELINE ADVANTAGES:
  ✓ Consistent and predictable results
  ✓ Fast processing (no analysis overhead)
  ✓ Simple implementation and maintenance
  ✓ Reliable baseline performance
  ✓ Particularly effective at: brightness_correction, contrast_enhancement

STATIC PIPELINE LIMITATIONS:
  ✗ Cannot adapt to image-specific characteristics
  ✗ May over-process or under-process certain images
  ✗ Fixed parameters may not be optimal for all cases
  ✗ Limited ability to handle varying quality levels

ADAPTIVE PIPELINE ADVANTAGES:
  ✓ Tailored processing for each image
  ✓ Can handle diverse image quality levels
  ✓ Optimizes processing order and parameters
  ✓ Better theoretical potential for improvement
  ✓ Particularly effective at: sharpness_improvement, brightness_correction

ADAPTIVE PIPELINE LIMITATIONS:
  ✗ More complex implementation and maintenance
  ✗ Longer processing time due to analysis overhead

✗ Potential for inconsistent results across similar images
✗ Requires careful tuning of decision thresholds
✗ May make suboptimal decisions with limited training data

RECOMMENDATIONS:
🎯 Adaptive preprocessing shows 9 more wins
→ Recommend adaptive approach for quality-critical applications
→ Consider static approach for high-throughput scenarios
💡 Hybrid approach: Use adaptive for difficult cases, static for routine processing
💡 Monitor performance on larger, more diverse datasets

Discussion and Future Work

Challenge	**Solution**
Large variability in image brightness, contrast, and noise across devices	Designed metric-based categorization (e.g., brightness levels) to adapt processing
Over-processing clean or high-quality images	Introduced quality scoring and conditional skipping of unnecessary steps
Balancing noise reduction and detail preservation	Used bilateral filtering and adaptive Gaussian blur with tuned parameters
Amplifying noise during sharpening	Skipped sharpening when noise levels were high
Determining optimal processing order	Adjusted pipeline order dynamically (e.g., denoise before contrast)
Evaluating performance across diverse samples	Computed pre- and post-processing quality scores with key metrics

Potential Improvements & Next Steps

1. **Data-Driven Parameter Optimization**
    - Learn optimal gamma, CLAHE, and denoising settings using a meta-model trained on quality metrics.
2. **Region-Based Preprocessing**
    - Apply localized preprocessing to high-interest areas (e.g., enamel, root apex) using segmentation maps.
3. **Feedback Loop from Downstream AI Models**
    - Integrate model feedback (e.g., detection confidence) to refine preprocessing decisions dynamically.
4. **Edge-Aware Enhancements**
    - Use adaptive unsharp masking and Laplacian filtering that respects anatomical boundaries.
5. **Integration with Clinical Metadata**
    - Adjust preprocessing based on patient age, device manufacturer, or tooth location for added context.

Preprocessing Benefit	**Impact on AI Tasks**
Standardized brightness & contrast	Improves feature visibility → enhances caries detection & margin clarity

Preprocessing Benefit	Impact on AI Tasks
Reduced noise	Increases model precision in detecting fine details like bone loss or fractures
Adaptive sharpening	Enhances edge information → better segmentation of enamel, pulp, and lesions
Minimized artifacts	Reduces false positives in classification models
Balanced enhancement	Improves generalization across data from different clinics

Instructions

1. **Install dependencies**
   - pip install numpy opencv-python matplotlib pydicom scikit-image pandas
2. Run static preprocessing
3. Run adaptive algorithm based preprocessing
4. (optional) run adaptive ML based preprocessing
5. Run eval_res.py to get evaluation results and a comparison of the methods