Notebook Explanation

1 Preparing the Dataset

Code

```python
import glob
import argparse
import shutil


if __name__ == '__main__':
    inp_folder = '/kaggle/input/alpha-dent/AlphaDent' + '/'
    out_folder = '/kaggle/output/alpha-dent/AlphaDent_4_classes' + '/'

    shutil.copytree(inp_folder, out_folder)

    # Replace txt files
    txt_paths = glob.glob(out_folder + '**/*.txt', recursive=True)
    for txt_path in txt_paths:
        lines = open(txt_path).readlines()
        out = open(txt_path, 'w')
        for line in lines:
            if line[0] == '4' or line[0] == '5' or line[0] == '6' or
    line[0] == '7' or line[0] == '8':
                out.write('3' + line[1:])
            else:
                out.write(line)
        out.close()

    id_to_classes = {
        1: 'Abrasion',
        2: 'Filling',
        3: 'Crown',
        4: 'Caries',
    }

    # Create .yaml file
    out = open(out_folder + 'yolo_seg_train.yaml', 'w')
    out.write('path: {}\n'.format(out_folder))
    out.write('train: images/train\n')
    out.write('val: images/valid\n')
    out.write('names:\n')
    out.write('  0: Abrasion\n')
    out.write('  1: Filling\n')
    out.write('  2: Crown\n')
    out.write('  3: Caries\n')
    out.close()


print("Done")
```

Listing 1: Copying dataset and adjusting labels

Explanation

This section prepares the dataset for 4-class training by:

- Copying the original dataset into a new folder.
- Merging labels with IDs 4–8 into ID 3 (`Caries`).
- Writing a YOLO-compatible YAML file describing the dataset structure and class names.

2 Environment Setup and Dataset Splitting

Code and Explanation

1. Environment Setup

```python
import os
import sys
import time
import glob
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from tqdm.auto import tqdm
import yaml
import random
from PIL import Image
import shutil
import warnings
warnings.filterwarnings('ignore')

# Disable wandb
os.environ['WANDB_DISABLED'] = 'true'

# Set random seeds for reproducibility
random.seed(42)
np.random.seed(42)
```

Listing 2: Import libraries, disable wandb, set random seeds

Explanation: This section imports all required libraries, disables `wandb`, and fixes random seeds to ensure reproducibility.

—

2. PyTorch and Ultralytics Setup

```python
# Install required packages
print("Installing required packages...")
os.system('pip install -q ultralytics')

import torch
from ultralytics import YOLO

# Set deterministic behavior for PyTorch
torch.manual_seed(42)
if torch.cuda.is_available():
    torch.cuda.manual_seed(42)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

print(f'\nPyTorch Version: {torch.__version__}')
print(f'CUDA Available: {torch.cuda.is_available()}')
if torch.cuda.is_available():
    print(f'CUDA Device: {torch.cuda.get_device_name(0)}')
```

Listing 3: Install ultralytics, set PyTorch deterministic behavior

Explanation: Installs Ultralytics YOLO, sets deterministic settings in PyTorch, and prints environment details.

—

3. Dataset Paths and Directory Creation

```
# Define original paths
BASE_PATH = '/kaggle/output/alpha-dent/AlphaDent_4_classes'
ORIGINAL_TRAIN_IMAGES_PATH = f'{BASE_PATH}/images/train'
VALID_IMAGES_PATH = f'{BASE_PATH}/images/valid'
TEST_IMAGES_PATH = f'{BASE_PATH}/images/test'
ORIGINAL_TRAIN_LABELS_PATH = f'{BASE_PATH}/labels/train'
VALID_LABELS_PATH = f'{BASE_PATH}/labels/valid'

# Output paths
OUTPUT_DIR = '/kaggle/working/'
WEIGHTS_DIR = f'{OUTPUT_DIR}/weights'
os.makedirs(WEIGHTS_DIR, exist_ok=True)

# Create new dataset structure with 90/10 split
NEW_DATASET_PATH = f'{OUTPUT_DIR}/alphadent_90_10_split'
NEW_TRAIN_IMAGES_PATH = f'{NEW_DATASET_PATH}/images/train'
NEW_EVAL_IMAGES_PATH = f'{NEW_DATASET_PATH}/images/eval'
NEW_VALID_IMAGES_PATH = f'{NEW_DATASET_PATH}/images/valid'
NEW_TEST_IMAGES_PATH = f'{NEW_DATASET_PATH}/images/test'
NEW_TRAIN_LABELS_PATH = f'{NEW_DATASET_PATH}/labels/train'
NEW_EVAL_LABELS_PATH = f'{NEW_DATASET_PATH}/labels/eval'
NEW_VALID_LABELS_PATH = f'{NEW_DATASET_PATH}/labels/valid'

# Create directories
os.makedirs(NEW_TRAIN_IMAGES_PATH, exist_ok=True)
os.makedirs(NEW_EVAL_IMAGES_PATH, exist_ok=True)
os.makedirs(NEW_VALID_IMAGES_PATH, exist_ok=True)
os.makedirs(NEW_TEST_IMAGES_PATH, exist_ok=True)
os.makedirs(NEW_TRAIN_LABELS_PATH, exist_ok=True)
os.makedirs(NEW_EVAL_LABELS_PATH, exist_ok=True)
os.makedirs(NEW_VALID_LABELS_PATH, exist_ok=True)
```

Listing 4: Define paths and create dataset structure

Explanation: Defines the original dataset structure, output directory, and creates new folders for the 90/10 split.

—

4. Train/Eval Split and File Copying

```python
print("\n=== Creating 90/10 Train/Eval Split ===")

# Get all training images
original_train_images = sorted(glob.glob(f'{ORIGINAL_TRAIN_IMAGES_PATH}/*.jpg'))
print(f"Total original training images: {len(original_train_images)}")

# Shuffle and split into 90% train, 10% eval
random.shuffle(original_train_images)
split_idx = int(0.9 * len(original_train_images))
train_90_images = original_train_images[:split_idx]
eval_10_images = original_train_images[split_idx:]

print(f"90% for training: {len(train_90_images)}")
print(f"10% for evaluation: {len(eval_10_images)}")

def copy_files(image_list, dest_images_dir, dest_labels_dir, source_labels_dir, desc):
    """Copy images and corresponding labels to destination directories."""
    for img_path in tqdm(image_list, desc=desc):
        img_filename = os.path.basename(img_path)
        shutil.copy2(img_path, os.path.join(dest_images_dir, img_filename))

        label_filename = img_filename.replace('.jpg', '.txt')
        source_label_path = os.path.join(source_labels_dir, label_filename)
        dest_label_path = os.path.join(dest_labels_dir, label_filename)

        if os.path.exists(source_label_path):
            shutil.copy2(source_label_path, dest_label_path)
```

Listing 5: Split training images into 90/10 and copy files

Explanation: Splits the training images into 90% train and 10% eval sets, then defines a helper function to copy images and their labels.

—

5. Copy Datasets

```python
# Copy 90% training data
copy_files(train_90_images, NEW_TRAIN_IMAGES_PATH, NEW_TRAIN_LABELS_PATH,
           ORIGINAL_TRAIN_LABELS_PATH, "Copying 90% training data")

# Copy 10% evaluation data
copy_files(eval_10_images, NEW_EVAL_IMAGES_PATH, NEW_EVAL_LABELS_PATH,
           ORIGINAL_TRAIN_LABELS_PATH, "Copying 10% evaluation data")

# Copy original validation data (unchanged)
original_valid_images = glob.glob(f'{VALID_IMAGES_PATH}/*.jpg')
copy_files(original_valid_images, NEW_VALID_IMAGES_PATH, NEW_VALID_LABELS_PATH,
           VALID_LABELS_PATH, "Copying validation data")

# Copy test data (images only, no labels)
print("Copying test images...")
test_images = glob.glob(f'{TEST_IMAGES_PATH}/*.jpg')
for img_path in tqdm(test_images, desc="Copying test images"):
    img_filename = os.path.basename(img_path)
    shutil.copy2(img_path, os.path.join(NEW_TEST_IMAGES_PATH, img_filename))
```

Listing 6: Copy training, evaluation, validation, and test data

Explanation: Copies the datasets into the new structure: training, evaluation, validation, and test (images only).

—

6. YOLO Config Creation

```python
# Define class information
CLASS_INFO = {
    0: {'name': 'Abrasion', 'description': 'Teeth with mechanical wear of hard tissues'},
    1: {'name': 'Filling', 'description': 'Dental fillings of various types'},
    2: {'name': 'Crown', 'description': 'Dental crown (restoration)'},
    3: {'name': 'Caries', 'description': 'Caries in fissures and pits'}
}

# Create YAML configuration for YOLO
print("\n=== Creating YOLO Configuration ===")
yolo_config = {
    'path': NEW_DATASET_PATH,
    'train': 'images/train',
    'val': 'images/valid',
    'test': 'images/test',
    'nc': 4,
    'names': [CLASS_INFO[i]['name'] for i in range(4)]
}

# Save the configuration
CUSTOM_YAML_PATH = f'{OUTPUT_DIR}/alphadent_config_90_10.yaml'
with open(CUSTOM_YAML_PATH, 'w') as f:
    yaml.dump(yolo_config, f, default_flow_style=False)
print(f"Created custom YAML config at: {CUSTOM_YAML_PATH}")
```

Listing 7: Create YOLO config and dataset stats

Explanation: Defines 4 dental classes, creates a YOLO config file, and saves it in YAML format.

—

7. Class Distribution Analysis

```
def analyze_class_distribution(labels_path, dataset_name):
    class_counts = {i: 0 for i in range(4)}
    total_annotations = 0
    label_files = glob.glob(f'{labels_path}/*.txt')

    for label_file in tqdm(label_files, desc=f"Analyzing {dataset_name}
    labels", leave=False):
        if os.path.exists(label_file) and os.path.getsize(label_file) >
    0:
            with open(label_file, 'r') as f:
                for line in f:
                    if line.strip():
                        class_id = int(line.strip().split()[0])
                        if 0 <= class_id < 4:
                            class_counts[class_id] += 1
                            total_annotations += 1
    return class_counts, total_annotations
```

Listing 8: Analyze class distribution in datasets

Explanation: Defines a function to count class occurrences in label files, helping check dataset balance.

—

8. Training Configuration and Start

```python
# Training configuration
print("\n=== Model Training Configuration ===")
EPOCHS = 30
IMAGE_SIZE = 640
BATCH_SIZE = 8 if torch.cuda.is_available() else 4
PATIENCE = 5

# Initialize and train model
print("\n=== Starting Model Training ===")
model = YOLO('yolov8x-seg.pt')

results = model.train(
    data=CUSTOM_YAML_PATH,
    epochs=EPOCHS,
    imgsz=IMAGE_SIZE,
    batch=BATCH_SIZE,
    patience=PATIENCE,
    save=True,
    save_period=10,
    project=OUTPUT_DIR,
    name='alphadent_yolov8x_90_10',
    exist_ok=True,
    pretrained=True,
    optimizer='AdamW',
    lr0=0.001,
    lrf=0.01,
    momentum=0.937,
    weight_decay=0.0005,
    warmup_epochs=3.0,
    warmup_momentum=0.8,
    warmup_bias_lr=0.1,
    box=7.5,
    cls=0.5,
    dfl=1.5,
    hsv_h=0.015,
    hsv_s=0.7,
    hsv_v=0.4,
    degrees=0.0,
    translate=0.1,
    scale=0.5,
    shear=0.0,
    perspective=0.0,
    flipud=0.0,
    fliplr=0.5,
    mosaic=1.0,
    mixup=0.0,
    copy_paste=0.0,
    plots=True,
    device=0 if torch.cuda.is_available() else 'cpu',
    workers=2,
    verbose=True,
    amp=True,
    val=True
)

```

```python
print("\nTraining completed!")
```

Listing 9: Training configuration and YOLOv8 training

Explanation: Defines hyperparameters (epochs, batch size, patience), initializes YOLOv8, and starts training with augmentation and optimization settings.

3 Model Loading, Validation and Evaluation

Code

```python
# Load best model
print("=== Loading Best Model ===")
best_model_path = f'{OUTPUT_DIR}/alphadent_yolov8x_90_10/weights/best.
    pt'
if os.path.exists(best_model_path):
    model = YOLO(best_model_path)
    print(f"Loaded best model from: {best_model_path}")
else:
    last_model_path = f'{OUTPUT_DIR}/alphadent_yolov8x_90_10/weights/
    last.pt'
    if os.path.exists(last_model_path):
        model = YOLO(last_model_path)
        print(f"Loaded last model from: {last_model_path}")
    else:
        print("Warning: No trained model found, using pretrained model"
    )
        model = YOLO('yolov8x-seg.pt')

# Validate model on original validation set
print("\n=== Model Validation on Original Validation Set ===")
try:
    metrics = model.val(
        data=CUSTOM_YAML_PATH,
        imgsz=IMAGE_SIZE,
        batch=1,
        conf=0.001,
        iou=0.5,
        max_det=300,
        device=0 if torch.cuda.is_available() else 'cpu',
        plots=False,
        save_json=False,
    )

    print(f"\nValidation Results on Original Validation Set:")
    print(f"mAP@50: {metrics.seg.map50:.4f}")
    print(f"mAP@50-95: {metrics.seg.map:.4f}")
except Exception as e:
    print(f"Validation error (non-critical): {e}")

# Create custom YAML for evaluation on the 10% held-out data
eval_config = {
    'path': NEW_DATASET_PATH,
    'train': 'images/train',
    'val': 'images/eval',  # Point to eval set for validation
    'test': 'images/test',
    'nc': 4,
```

```
    'names': [CLASS_INFO[i]['name'] for i in range(4)]
}

EVAL_YAML_PATH = f'{OUTPUT_DIR}/alphadent_eval_config.yaml'
with open(EVAL_YAML_PATH, 'w') as f:
    yaml.dump(eval_config, f, default_flow_style=False)

# Evaluate model on the 10% held-out labeled data
print("\n=== Model Evaluation on 10% Held-out Labeled Data ===")
try:
    eval_metrics = model.val(
        data=EVAL_YAML_PATH,
        imgsz=IMAGE_SIZE,
        batch=1,
        conf=0.001,
        iou=0.5,
        max_det=300,
        device=0 if torch.cuda.is_available() else 'cpu',
        plots=True,
        save_json=True,
        name='eval_10_percent'
    )

    print(f"\nEvaluation Results on 10% Held-out Data:")
    print(f"mAP@50: {eval_metrics.seg.map50:.4f}")
    print(f"mAP@50-95: {eval_metrics.seg.map:.4f}")
    print(f"Per-class mAP@50:")
    for i, map_val in enumerate(eval_metrics.seg.maps):
        print(f"  {CLASS_INFO[i]['name']}: {map_val:.4f}")

except Exception as e:
    print(f"Evaluation error: {e}")

# Final summary
print("\n=== Summary ===")
print(f"    Successfully created 90/10 train/eval split")
print(f"    Trained model on 90% of original training data ({len(new_train_images)} images)")
print(f"    Evaluated model on 10% held-out labeled data ({len(new_eval_images)} images)")
print(f"    Also validated on original validation set ({len(new_valid_images)} images)")
print(f"    Model weights saved to: {OUTPUT_DIR}/alphadent_yolov8x_90_10/weights/")
```

Listing 10: Load best model, validate and evaluate on held-out data

Explanation

- Loads the best available checkpoint (best → last → pretrained) and runs validation.
- Validates on both the original validation set and the 10
- Writes a small eval YAML pointing to the held-out eval split.
- Prints a final summary of dataset sizes and where the model weights are stored.

4 Inference and Submission Pipeline

Code and Explanation

1. Clear Cache and Define Converter Function

```python
import torch
torch.cuda.empty_cache()
# Inference on test set
print("\n=== Running Inference on Test Set ===")

def convert_to_submission_format(results, image_paths):
    submission_rows = []

    for idx, result in enumerate(results):
        # Get image ID
        image_id = os.path.basename(image_paths[idx]).replace('.jpg', '')

        if result.masks is not None and len(result.masks) > 0:
            try:
                masks = result.masks.xy
                classes = result.boxes.cls.cpu().numpy().astype(int)
                confidences = result.boxes.conf.cpu().numpy()
                h, w = result.orig_shape

                # Process each detection
                for mask_idx in range(len(masks)):
                    if mask_idx < len(classes) and mask_idx < len(confidences):
                        polygon = masks[mask_idx]

                        if len(polygon) >= 3:  # Valid polygon
                            normalized_coords = []
                            for point in polygon:
                                x_norm = float(point[0]) / w
                                y_norm = float(point[1]) / h
                                x_norm = max(0.0, min(1.0, x_norm))
                                y_norm = max(0.0, min(1.0, y_norm))
                                normalized_coords.extend([x_norm, y_norm])

                            poly_str = ' '.join([f'{coord:.6f}' for coord in normalized_coords])

                            submission_rows.append({
                                'patient_id': image_id,
                                'class_id': int(classes[mask_idx]),
                                'confidence': float(confidences[mask_idx]),
                                'poly': poly_str
                            })
            except Exception as e:
                print(f"Error processing result for image {idx}: {e}")
                continue

```

```
    return submission_rows
```

Listing 11: Empty CUDA cache and define conversion function

Explanation: First, GPU cache is cleared to avoid memory issues. Then, a function is defined to **convert YOLO results into Kaggle submission format**: extracting polygons, normalizing coordinates to $[0, 1]$, and storing predictions as rows.

—

2. Run Inference on Test Images

```
# Process test images
test_images = sorted(glob.glob(f'{TEST_IMAGES_PATH}/*.jpg'))
all_submission_rows = []
INFERENCE_BATCH_SIZE = 2 if torch.cuda.is_available() else 4

print(f"Processing {len(test_images)} test images...")

for i in tqdm(range(0, len(test_images), INFERENCE_BATCH_SIZE)):
    torch.cuda.empty_cache()
    batch_images = test_images[i:i + INFERENCE_BATCH_SIZE]

    try:
        results = model.predict(
            batch_images,
            imgsz=IMAGE_SIZE,
            conf=0.25,
            iou=0.45,
            max_det=300,
            device=0 if torch.cuda.is_available() else 'cpu',
            verbose=False,
            agnostic_nms=True,
            retina_masks=True,
        )

        batch_rows = convert_to_submission_format(results, batch_images)
        all_submission_rows.extend(batch_rows)

    except Exception as e:
        print(f"Error in batch {i//INFERENCE_BATCH_SIZE}: {e}")
        continue

print(f"\nGenerated {len(all_submission_rows)} predictions")
```

Listing 12: Process test set images in batches

Explanation: All test images are processed in **small batches** to avoid GPU memory overflow. The predictions are generated with YOLO, filtered by confidence (0.25) and IoU (0.45), then converted into submission rows using the function defined earlier.

—

3. Create Submission DataFrame

```python
# Create submission DataFrame
print("\n=== Creating Submission File ===")
submission_df = pd.DataFrame(all_submission_rows)

# Ensure all test images are included
all_test_ids = [os.path.basename(img).replace('.jpg', '') for img in test_images]
if len(submission_df) > 0:
    predicted_ids = submission_df['patient_id'].unique()
    missing_ids = set(all_test_ids) - set(predicted_ids)
else:
    missing_ids = set(all_test_ids)

# Add dummy predictions if missing
if missing_ids:
    print(f"Adding dummy predictions for {len(missing_ids)} images without detections")
    dummy_rows = []
    for img_id in missing_ids:
        dummy_rows.append({
            'patient_id': img_id,
            'class_id': 0,
            'confidence': 0.01,
            'poly': '0.1 0.1 0.1 0.2 0.2 0.2 0.2 0.1'
        })

    submission_df = pd.concat([submission_df, pd.DataFrame(dummy_rows)], ignore_index=True)

submission_df = submission_df.sort_values(['patient_id', 'confidence'], ascending=[True, False])
submission_df = submission_df[['patient_id', 'class_id', 'confidence', 'poly']]
submission_df.to_csv('submission.csv', index=False)
print("Main submission file created: submission.csv")
```

Listing 13: Build submission file from predictions

Explanation: A DataFrame is created for submission. If some images have no detections, **dummy polygons** with low confidence are added to satisfy Kaggle format requirements. Finally, predictions are sorted and saved into `submission.csv`.

—

4. Verification of Submission

```python
# Verify submission format
print("\n=== Verifying Submission Format ===")
print(f"Total predictions: {len(submission_df)}")
print(f"Unique images: {submission_df['patient_id'].nunique()}")
print(f"All test images included: {submission_df['patient_id'].nunique() == len(test_images)}")

print("\nFirst 5 rows of submission:")
print(submission_df.head())

# Check issues
print("\n=== Checking for Potential Issues ===")
missing_in_submission = set(all_test_ids) - set(submission_df['patient_id'].unique())
if missing_in_submission:
    print(f"WARNING: Missing images in submission: {missing_in_submission}")
else:
    print("    All test images have predictions")

# Class distribution
print("\nPredictions per class:")
class_dist = submission_df['class_id'].value_counts().sort_index()
for class_id, count in class_dist.items():
    if 0 <= class_id < 9:
        print(f"  Class {class_id} ({CLASS_INFO[class_id]['name']}): {count}")

# Confidence statistics
print(f"\nConfidence statistics:")
print(f"  Min: {submission_df['confidence'].min():.4f}")
print(f"  Max: {submission_df['confidence'].max():.4f}")
print(f"  Mean: {submission_df['confidence'].mean():.4f}")
print(f"  Median: {submission_df['confidence'].median():.4f}")
```

Listing 14: Verify submission consistency

Explanation: Ensures every test image has predictions, prints class distribution, and summarizes confidence values to check prediction quality.

—

5. High-Confidence Alternative Submission

```python
# Create alternative submission with higher confidence threshold
print("\n=== Creating Alternative Submission (Higher Confidence) ===")
high_conf_df = submission_df[submission_df['confidence'] >= 0.3].copy()

# Ensure coverage of all images
high_conf_ids = high_conf_df['patient_id'].unique()
missing_high_conf = set(all_test_ids) - set(high_conf_ids)

if missing_high_conf:
    for img_id in missing_high_conf:
        img_preds = submission_df[submission_df['patient_id'] == img_id
    ]
        if len(img_preds) > 0:
            high_conf_df = pd.concat([high_conf_df, img_preds.head(1)],
     ignore_index=True)
        else:
            dummy_row = pd.DataFrame([{
                'patient_id': img_id,
                'class_id': 0,
                'confidence': 0.01,
                'poly': '0.1 0.1 0.1 0.2 0.2 0.2 0.2 0.1'
            }])
            high_conf_df = pd.concat([high_conf_df, dummy_row],
    ignore_index=True)

high_conf_df = high_conf_df.sort_values(['patient_id', 'confidence'],
    ascending=[True, False])
high_conf_df.insert(0, 'id', range(1, len(high_conf_df) + 1))
high_conf_df.to_csv('submission_high_conf.csv', index=False)
print(f"Created high confidence submission with {len(high_conf_df)}
    predictions")

print("\n=== Pipeline Completed Successfully! ===")
print("Submission files created:")
print("  - submission.csv (main submission)")
print("  - submission_high_conf.csv (alternative with higher confidence
     threshold)")
```

Listing 15: Generate high-confidence version of submission

Explanation: An additional submission file is created with a **confidence threshold of 0.3**. If some images are missing predictions at this threshold, the highest-confidence detection (or a dummy polygon) is added back. This produces `submission_high_conf.csv`, a stricter version of the submission.

5 Polygon-based Caries Classification Using ResNet18

This section describes the implementation of a polygon-based caries classifier built on top of a pre-trained ResNet18 model. The pipeline includes dataset preparation, model training, and evaluation.

5.1 Environment and Settings

We begin by importing the required libraries and setting paths, hyperparameters, and device configuration:

```python
import os
import cv2
import torch
import numpy as np
from torch import nn
from torch.utils.data import Dataset, DataLoader
from torchvision import models, transforms
from PIL import Image
from glob import glob

# Paths and settings
TRAIN_PATH = "/kaggle/input/alpha-dent/AlphaDent/images/train"
TRAIN_LABEL_PATH = "/kaggle/input/alpha-dent/AlphaDent/labels/train"
NUM_CLASSES = 6   # caries1 -> caries6
IMG_SIZE = 224    # ResNet input size
BATCH_SIZE = 16
EPOCHS = 5
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
```

We define six caries sub-classes, set image size for ResNet input, batch size, and number of epochs. Computation runs on GPU if available.

5.2 Custom Dataset: Polygon Cropping

A custom `Dataset` class is implemented to load images and extract polygon regions corresponding to caries annotations:

```python
class PolygonCariesDataset(Dataset):
    def __init__(self, image_dir, label_dir, transform=None):
        self.samples = []
        self.transform = transform

        image_paths = sorted(glob(os.path.join(image_dir, "*.jpg")))
        label_paths = sorted(glob(os.path.join(label_dir, "*.txt")))

        for img_path, lbl_path in zip(image_paths, label_paths):
            with open(lbl_path, "r") as f:
                lines = f.readlines()
            for line in lines:
                parts = line.strip().split()
                class_id = int(parts[0])
                if 3 <= class_id <= 8:  # merged caries classes
                    coords = np.array(parts[1:], dtype=float).reshape(-1, 2)
                    self.samples.append((img_path, coords, class_id-3))

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        img_path, coords, label = self.samples[idx]
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```python
        h, w, _ = img.shape

        # Crop polygon bounding box
        xs, ys = coords[:,0], coords[:,1]
        x1, y1 = max(int(xs.min()*w),0), max(int(ys.min()*h),0)
        x2, y2 = min(int(xs.max()*w), w-1), min(int(ys.max()*h), h-1)
        crop = img[y1:y2, x1:x2]

        # Transform to tensor
        crop = Image.fromarray(crop).convert("RGB")
        if self.transform:
            crop = self.transform(crop)
        return crop, label
```

This class reads YOLO-format labels, filters only caries classes, crops bounding boxes from polygon masks, and returns cropped image patches with labels.

5.3 Data Transforms and Loader

The cropped patches are resized and normalized before being loaded into a `DataLoader`:

```python
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

train_dataset = PolygonCariesDataset(TRAIN_PATH, TRAIN_LABEL_PATH,
    transform=transform)
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle
    =True)
```

5.4 Model: ResNet18 Fine-tuning

A pre-trained ResNet18 is adapted by replacing its final fully-connected layer with a classifier for six caries classes:

```python
model = models.resnet18(pretrained=True)
model.fc = nn.Linear(model.fc.in_features, NUM_CLASSES)
model = model.to(DEVICE)
```

5.5 Training Setup

We use cross-entropy loss and Adam optimizer:

```python
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
```

5.6 Training Loop

The model is trained for 5 epochs, reporting loss and accuracy per epoch:

```python
for epoch in range(EPOCHS):
    model.train()
    running_loss, correct, total = 0.0, 0, 0

```

```python
    for imgs, labels in train_loader:
        imgs, labels = imgs.to(DEVICE), labels.to(DEVICE)

        optimizer.zero_grad()
        outputs = model(imgs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print(f"Epoch [{epoch+1}/{EPOCHS}] "
          f"Loss: {running_loss/len(train_loader):.4f} "
          f"Accuracy: {correct/total:.4f}")
```

5.7 Saving the Model

After training, the model weights are saved:

```python
torch.save(model.state_dict(), "caries_polygon_classifier.pth")
```

5.8 Validation and Metrics

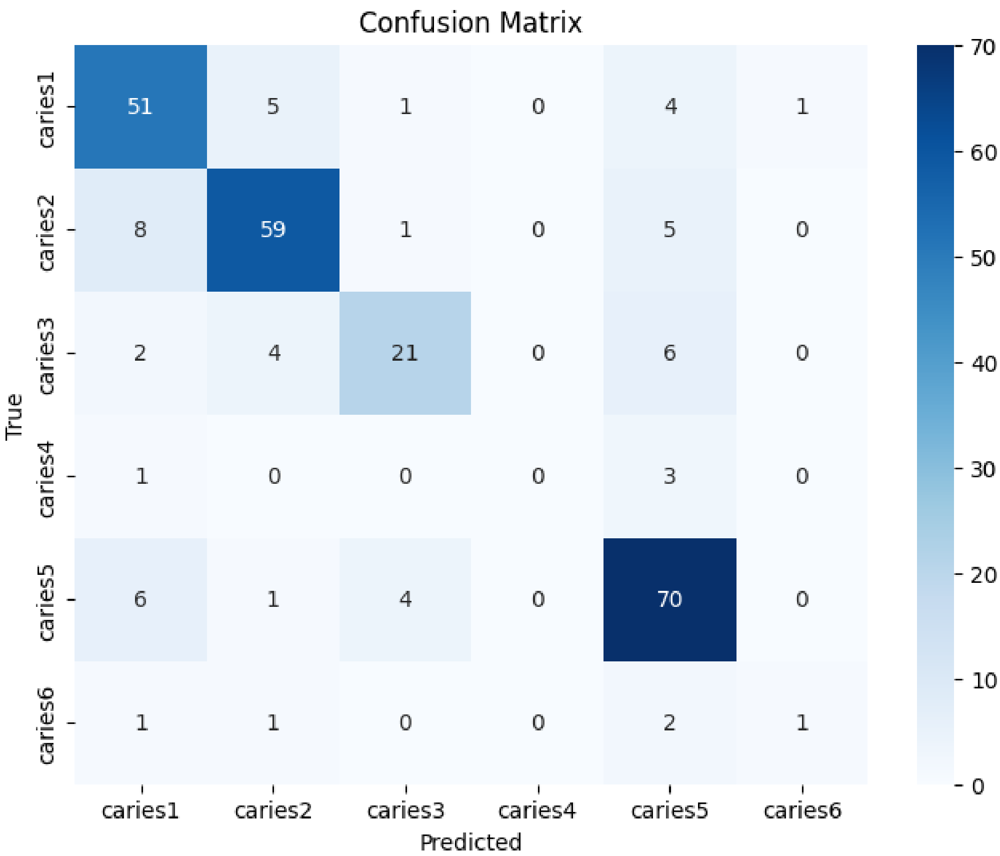
The trained model is evaluated on a validation set, computing classification report and confusion matrix:

```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

val_dataset = PolygonCariesDataset(VAL_PATH, VAL_LABEL_PATH, transform=
    transform)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=
    False)

model.eval()
all_labels, all_preds = [], []
with torch.no_grad():
    for imgs, labels in val_loader:
        imgs, labels = imgs.to(DEVICE), labels.to(DEVICE)
        outputs = model(imgs)
        _, preds = outputs.max(1)
        all_labels.extend(labels.cpu().numpy())
        all_preds.extend(preds.cpu().numpy())

print(classification_report(all_labels, all_preds,
      target_names=[f"caries{i}" for i in range(1, NUM_CLASSES+1)]))

cm = confusion_matrix(all_labels, all_preds)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=[f"caries{i}" for i in range(1, NUM_CLASSES+1)
    ],
            yticklabels=[f"caries{i}" for i in range(1, NUM_CLASSES+1)
    ])
```

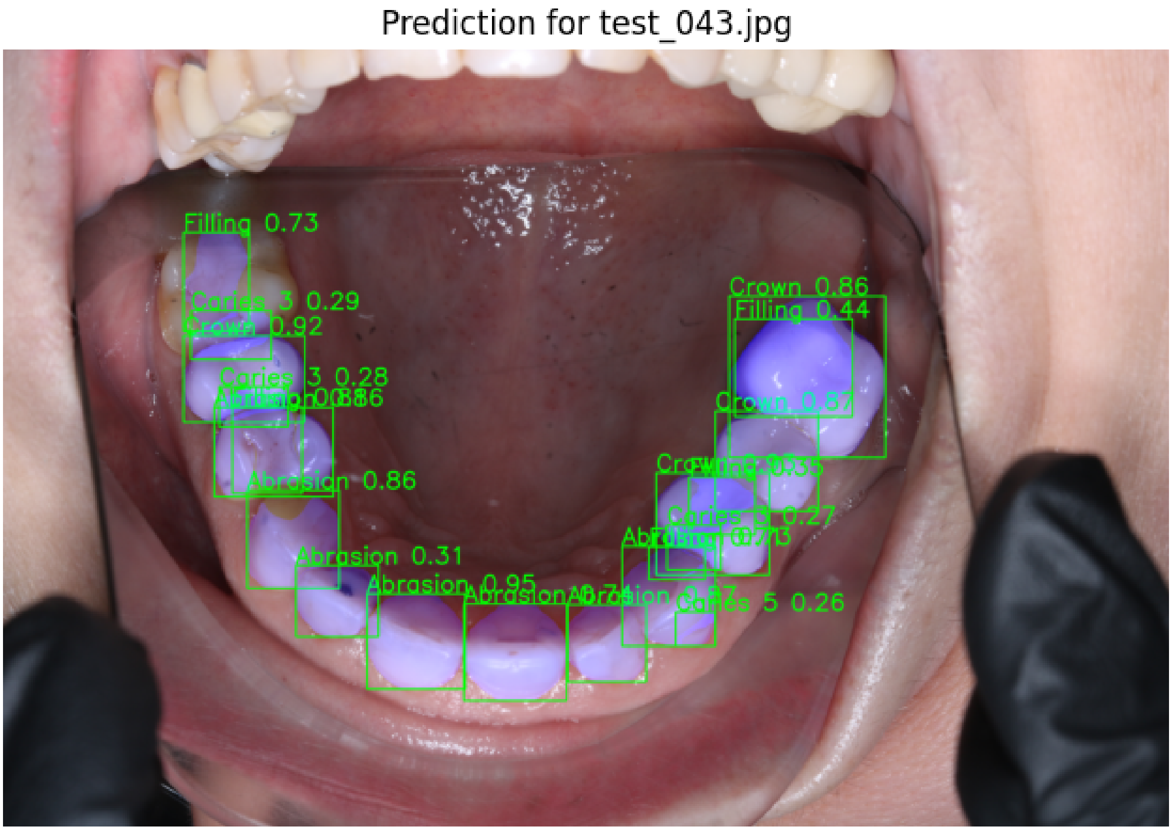
```python
plt.title("Confusion Matrix")
plt.show()
```

Table 1: Classification Report for Caries Classification

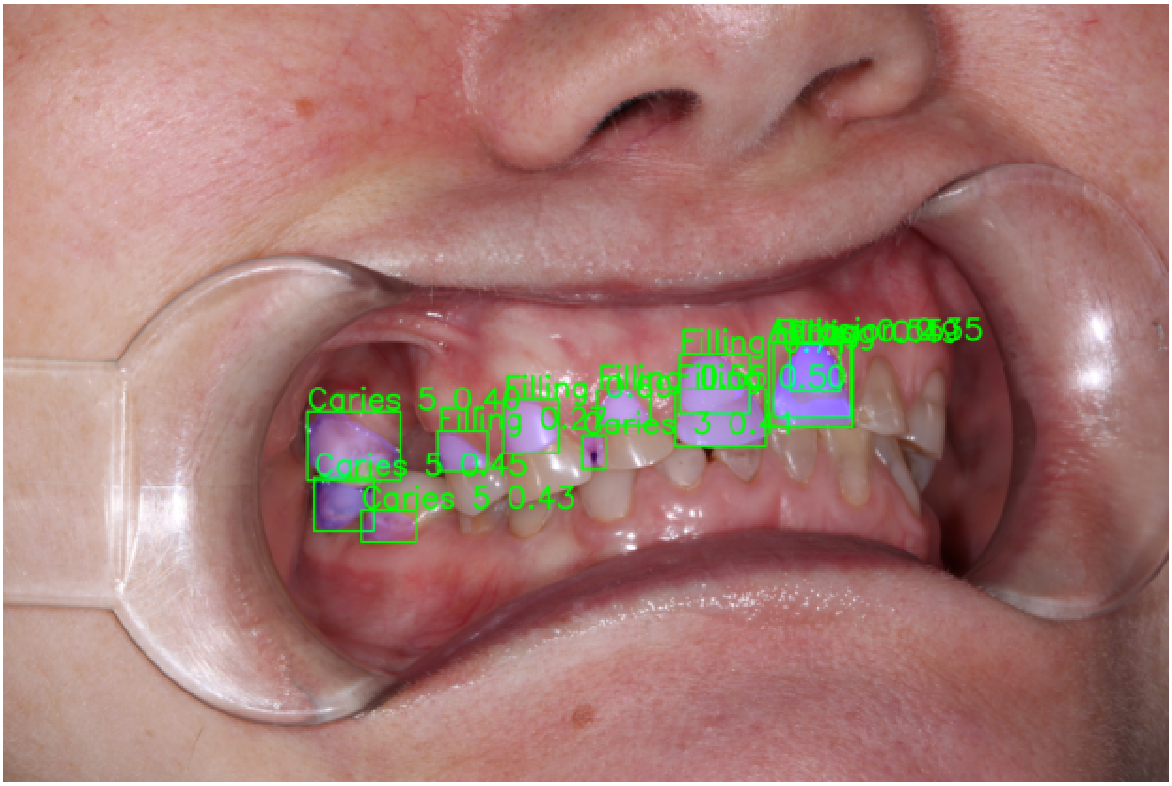
Class	**Precision**	**Recall**	**F1-Score**	**Support**
Caries1	0.74	0.82	0.78	62
Caries2	0.84	0.81	0.83	73
Caries3	0.78	0.64	0.70	33
Caries4	0.00	0.00	0.00	4
Caries5	0.78	0.86	0.82	81
Caries6	0.50	0.20	0.29	5
Accuracy			0.78	258
Macro Avg	0.61	0.56	0.57	258
Weighted Avg	0.77	0.78	0.77	258



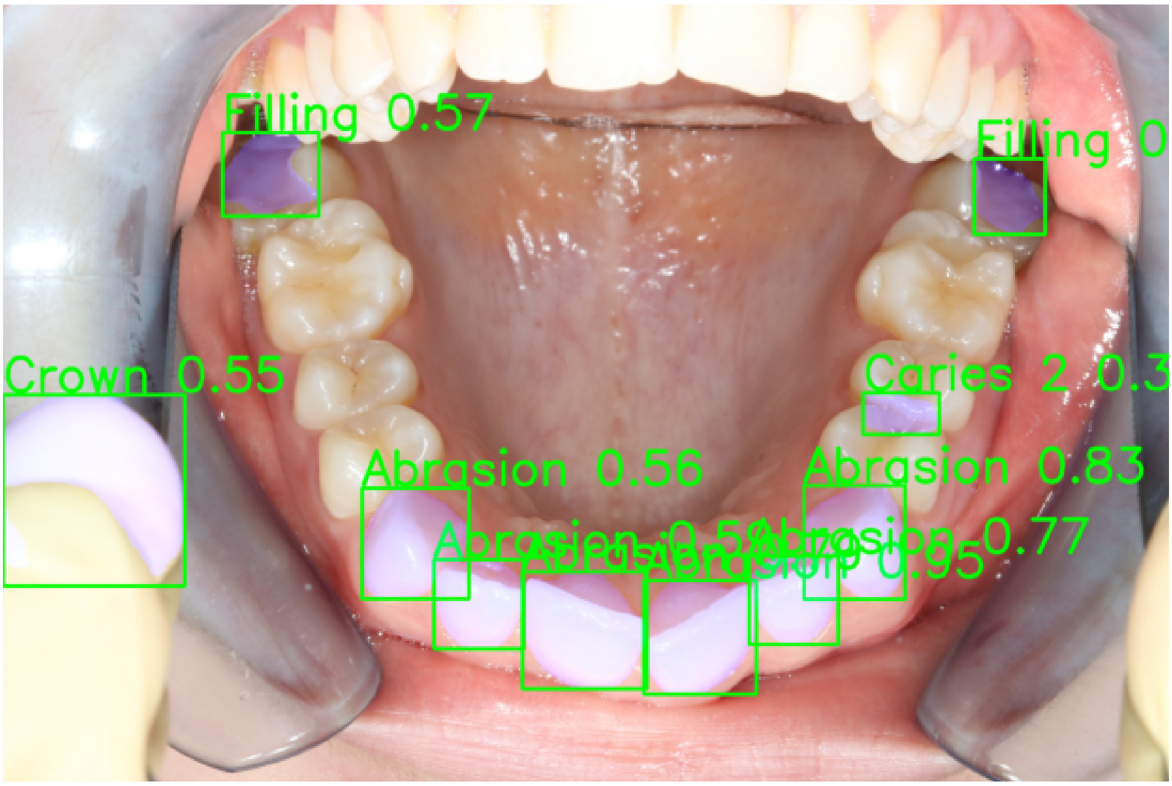
6 Classification Examples



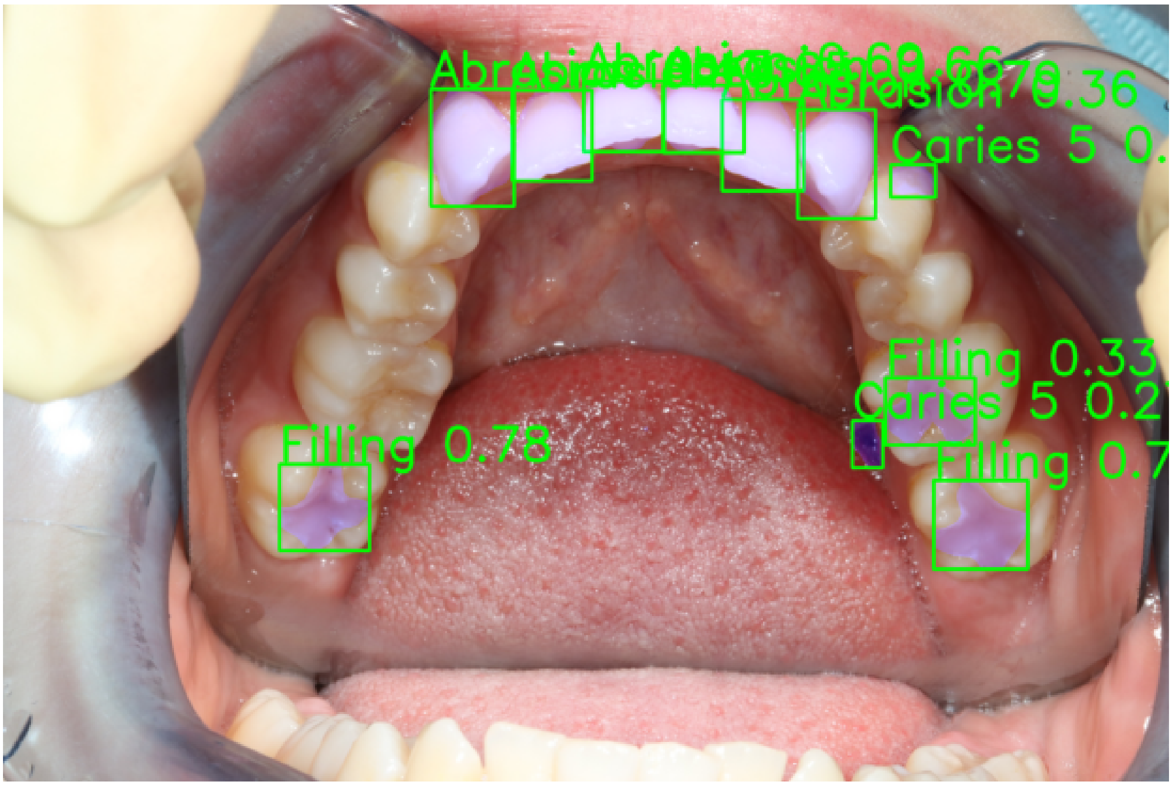
Prediction for test_052.jpg



Prediction for test_040.jpg



Prediction for test_041.jpg



Prediction for test_014.jpg