Project: Automated Bruise Detection System using YOLOv8

In the fields of medical diagnosis and forensic examination, the accurate assessment of soft tissue injuries, such as bruises, plays a pivotal role; however, this process often relies heavily on subjective visual observation and can be easily affected by environmental conditions. This project aims to automate this procedure by developing a high-speed Computer Vision model capable of detecting and localizing bruises with precision. We utilize the **YOLOv8** architecture (State-of-the-art) and apply **Transfer Learning** techniques on the `yolov8n` (Nano) version to fine-tune the model using the specialized `Bruise-Detection-6` dataset from Roboflow. This notebook presents the complete End-to-End experimental workflow - from environment setup and data preparation to training the feature extraction model - and concludes with an inference deployment to detect bruises in real-world video footage, demonstrating the feasibility of applying AI to support injury diagnosis.

1. Environment Setup and Dataset Preparation

In this step, we perform the necessary setup to prepare the environment for training:

1. Mount Google Drive: We mount the drive to `/content/drive` to ensure that our training logs and saved models are stored persistently and not lost when the Colab session ends.

```
from google.colab import drive
drive.mount('/content/drive')
```

1. Install Roboflow: We install the Roboflow library to manage our dataset.
2. Download Dataset: We authenticate with the Roboflow API and download the `Bruise-Detection-6` dataset. The dataset is automatically formatted for **YOLOv8** (folders for train/validation images and labels).

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="0zp5QxQsjebgT2hYoeIM")
project = rf.workspace("dominic-antigua").project("bruise-detection")
version = project.version(6)
dataset = version.download("yolov8")
```

2. Training the YOLOv8 Model

This section handles the core training process using the Ultralytics library.

- Install Ultralytics: We install the official YOLOv8 library.

```
pip install ultralytics
```

- Load Model: We initialize the model using weights from `yolov8n.pt` (Nano version). This uses **transfer learning** to speed up training compared to starting from scratch.
- Train: We execute the training loop with the following key parameters:
  - `data`: Points to the `data.yaml` file from our downloaded dataset.
  - `epochs=100`: The model iterates through the entire dataset 100 times.
  - `imgsz=640`: The input images are resized to 640x640 pixels.
  - `project`: We specify a path in Google Drive (`YOLOv8_Bruise_Detection_Models`) to save the training results and model weights automatically.

```
from ultralytics import YOLO

# Load a pre-trained YOLO model (e.g., yolov8n.pt for nano version)
model = YOLO('yolov8n.pt') # You can choose other YOLOv8 models like
'yolov8s.pt', 'yolov8m.pt', etc.

# Train the model using the downloaded dataset. The 'data' argument
points to the dataset's 'data.yaml' file.
# Assuming Roboflow downloaded the dataset into a directory named
'collision-1'
# The data.yaml file should be inside this directory.
results = model.train(data='/content/Bruise-Detection-6/data.yaml',
epochs=100, imgsz=640,

project='/content/drive/MyDrive/YOLOv8_Bruise_Detection_Models', #
Specify project directory on Google Drive
                      name='bruise_detection_run') # A name for the
training run

print("Training completed. The 'best.pt' model will be saved in the
'runs/detect/train/weights' directory.")
print("Also, it will be saved to your Google Drive at
/content/drive/MyDrive/YOLOv8_Bruise_Detection_Models/bruise_detection
_run/weights/best.pt")
```

3. Video Inference and Processing

After training, we use the trained model to detect bruises on a new video file (`Bruise3.mp4`).

1. *Load Custom Weights*: We load the best-performing weights (`best.pt`) saved during the training phase from Google Drive.
2. *Video Processing*: We use **OpenCV** to read the input video frame by frame.
3. *Detection Loop*:
   - For each frame, the model predicts objects with a *confidence threshold of 0.7* (only high-confidence detections are shown).
   - The `plot()` function draws bounding boxes and labels around detected bruises.
4. *Save Output*: The annotated frames are compiled into a new video file named `processed_output_video.mp4`.

```
import cv2
from ultralytics import YOLO

# Load the trained YOLOv8 model
model =
YOLO('/content/drive/MyDrive/YOLOv8_Bruise_Detection_Models/bruise_det
ection_run/weights/best.pt')

# Define the path to your input video file
input_video_path = "/content/drive/MyDrive/Bruise3.mp4"

# Define the desired name for the output video file
output_video_path = "processed_output_video.mp4" # Using a new name to
avoid conflicts

# Open the input video file
cap = cv2.VideoCapture(input_video_path)

# Check if video opened successfully
if not cap.isOpened():
    print(f"Error: Could not open video file {input_video_path}")
else:
    # Get video properties
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)

    # Define the codec and create VideoWriter object
    # Use 'mp4v' for .mp4 files. Other codecs might be 'XVID', 'MJPG',
etc.
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_video_path, fourcc, fps,
(frame_width, frame_height))

    print(f"Processing video: {input_video_path}")
    print(f"Output will be saved to: {output_video_path}")

    frame_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Perform object detection on the frame (adjust conf as
needed, e.g., 0.25 is default)
        results = model(frame, conf=0.7) # Using a confidence of 0.7
as per previous instruction

        # Draw bounding boxes and labels on the frame
        annotated_frame = results[0].plot()
```

```
        # Write the processed frame to the output video
        out.write(annotated_frame)
        frame_count += 1
        if frame_count % 100 == 0:
            print(f"Processed {frame_count} frames...")

    # Release everything when job is finished
    cap.release()
    out.release()
    print("Video processing complete. The processed video is saved as
'" + output_video_path + "'.")
```