Dataset Loader Fix

```
import os
import xml.etree.ElementTree as ET
from PIL import Image
from typing import List, Dict, Tuple
import torch
from torch.utils.data import Dataset
from torchvision import transforms


class CutInSequenceDataset(Dataset):
    def __init__(self, root_dir: str, sequence_length: int = 5, transform=None):
        self.root_dir = root_dir
        self.sequence_length = sequence_length
        self.transform = transform or transforms.Compose([
            transforms.Resize((512, 512)),
            transforms.ToTensor()
        ])
        self.samples = self._collect_sequences()

    def _collect_sequences(self) -> List[Tuple[str, str, List[str]]]:
        sequences = []
        for recording in sorted(os.listdir(self.root_dir)):
            rec_path = os.path.join(self.root_dir, recording)
            anno_dir = os.path.join(rec_path, 'Annotations')
            img_dir = os.path.join(rec_path, 'Images')
            if not os.path.isdir(anno_dir):
                continue

            frames = sorted([f for f in os.listdir(anno_dir) if f.endswith('.xml')])
            for i in range(len(frames) - self.sequence_length + 1):
                frame_seq = frames[i:i + self.sequence_length]
                sequences.append((recording, img_dir, frame_seq))
        return sequences

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

    def __getitem__(self, idx):
        recording, img_dir, frame_seq = self.samples[idx]
        images = []
        annotations = []

        for xml_file in frame_seq:
            frame_id = xml_file.replace('.xml', '')
            img_path = os.path.join(img_dir, f"{frame_id}.JPG")
            xml_path = os.path.join(self.root_dir, recording, 'Annotations', xml_file)

            image = Image.open(img_path).convert('RGB')
            image = self.transform(image)
            images.append(image)

            ann = self._parse_xml(xml_path)
            annotations.append(ann)

        return torch.stack(images), annotations

    def _parse_xml(self, xml_path: str) -> List[Dict]:
        objects = []
        tree = ET.parse(xml_path)
        root = tree.getroot()
        for obj in root.findall('object'):
            name = obj.find('name').text
            bbox = obj.find('bndbox')
            xmin = float(bbox.find('xmin').text)
            ymin = float(bbox.find('ymin').text)
            xmax = float(bbox.find('xmax').text)
            ymax = float(bbox.find('ymax').text)

            cutting = False
            for attr in obj.find('attributes').findall('attribute'):
                if attr.find('name').text == 'Cutting':
                    cutting = attr.find('value').text.lower() == 'true'

            objects.append({
                'label': name,
                'bbox': [xmin, ymin, xmax, ymax],
                'cutting': cutting
            })
        return objects
```

this is my dataset loader.

my dataset has following directory structure defined.

```
distribution
|
|--Train
   |
   |-- REC_2020_07_12_02_05_26_F
       |
       |-- Annotations/
            |- frame_000000.jpg
            |- frame_000000.xml
```

in this format so let's fix the path to not lookup for images and xml annotations separately rather they both files lie in Annotation directory only.

Thanks for the clarification. Since both the **images** and **XML annotation files** are in the same `Annotations/` directory, we can simplify the dataset loader by only referencing that one directory for both file types.

Here's your corrected version of `CutInSequenceDataset`:

✅ Updated Dataset Loader Code

```python
import os
import xml.etree.ElementTree as ET
from PIL import Image
from typing import List, Dict, Tuple
import torch
from torch.utils.data import Dataset
from torchvision import transforms


class CutInSequenceDataset(Dataset):
    def __init__(self, root_dir: str, sequence_length: int = 5, transform=None):
        self.root_dir = root_dir
        self.sequence_length = sequence_length
        self.transform = transform or transforms.Compose([
            transforms.Resize((512, 512)),
            transforms.ToTensor()
        ])
        self.samples = self._collect_sequences()

    def _collect_sequences(self) -> List[Tuple[str, str, List[str]]]:
        sequences = []
        for recording in sorted(os.listdir(self.root_dir)):
            anno_dir = os.path.join(self.root_dir, recording, 'Annotations')
            if not os.path.isdir(anno_dir):
                continue

            frames = sorted([f for f in os.listdir(anno_dir) if f.endswith('.xml')])
            for i in range(len(frames) - self.sequence_length + 1):
                frame_seq = frames[i:i + self.sequence_length]
                sequences.append((recording, anno_dir, frame_seq))
        return sequences

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

    def __getitem__(self, idx):
        recording, anno_dir, frame_seq = self.samples[idx]
        images = []
        annotations = []

        for xml_file in frame_seq:
            frame_id = xml_file.replace('.xml', '')
            img_path = os.path.join(anno_dir, f"{frame_id}.JPG")
            xml_path = os.path.join(anno_dir, xml_file)

            image = Image.open(img_path).convert('RGB')
            image = self.transform(image)
            images.append(image)

            ann = self._parse_xml(xml_path)
            annotations.append(ann)

        return torch.stack(images), annotations
    def _parse_xml(self, xml_path: str) -> List[Dict]:
        objects = []
        tree = ET.parse(xml_path)
        root = tree.getroot()
        for obj in root.findall('object'):
            name = obj.find('name').text
            bbox = obj.find('bndbox')
            xmin = float(bbox.find('xmin').text)
            ymin = float(bbox.find('ymin').text)
            xmax = float(bbox.find('xmax').text)
            ymax = float(bbox.find('ymax').text)

            cutting = False
            attributes = obj.find('attributes')
            if attributes is not None:
                for attr in attributes.findall('attribute'):
                    if attr.find('name').text == 'Cutting':
                        cutting = attr.find('value').text.lower() == 'true'

            objects.append({
                'label': name,
                'bbox': [xmin, ymin, xmax, ymax],
                'cutting': cutting
            })
        return objects
```

🔧 Key Fixes:

- Only uses `Annotations/` folder now for both images (`.JPG`) and labels (`.xml`).
- Proper fallback if `<attributes>` is missing (optional in some frames).
- More readable and error-resilient path generation.

Let me know if you want this extended to support frame skipping, augmentation, or training tag filtering (`cutting=True` vs `False`).

now provide code to run in notebook

Here's the **full notebook code** to use your `CutInSequenceDataset` and test it step-by-step in a Colab/enterprise environment.

✅ 1. Import and Dataset Loader

```
python

import os
import xml.etree.ElementTree as ET
from PIL import Image
from typing import List, Dict, Tuple
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

✅ 2. Define the Custom Dataset Class

```python
class CutInSequenceDataset(Dataset):
    def __init__(self, root_dir: str, sequence_length: int = 5, transform=None):
        self.root_dir = root_dir
        self.sequence_length = sequence_length
        self.transform = transform or transforms.Compose([
            transforms.Resize((512, 512)),
            transforms.ToTensor()
        ])
        self.samples = self._collect_sequences()

    def _collect_sequences(self) -> List[Tuple[str, str, List[str]]]:
        sequences = []
        for recording in sorted(os.listdir(self.root_dir)):
            anno_dir = os.path.join(self.root_dir, recording, 'Annotations')
            if not os.path.isdir(anno_dir):
                continue

            frames = sorted([f for f in os.listdir(anno_dir) if f.endswith('.xml')])
            for i in range(len(frames) - self.sequence_length + 1):
                frame_seq = frames[i:i + self.sequence_length]
                sequences.append((recording, anno_dir, frame_seq))
        return sequences

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

    def __getitem__(self, idx):
        recording, anno_dir, frame_seq = self.samples[idx]
        images = []
        annotations = []

        for xml_file in frame_seq:
            frame_id = xml_file.replace('.xml', '')
            img_path = os.path.join(anno_dir, f"{frame_id}.JPG")
            xml_path = os.path.join(anno_dir, xml_file)

            image = Image.open(img_path).convert('RGB')
            image = self.transform(image)
            images.append(image)

            ann = self._parse_xml(xml_path)
            annotations.append(ann)

        return torch.stack(images), annotations

    def _parse_xml(self, xml_path: str) -> List[Dict]:
        objects = []
        tree = ET.parse(xml_path)
        root = tree.getroot()
        for obj in root.findall('object'):
            name = obj.find('name').text
            bbox = obj.find('bndbox')
            xmin = float(bbox.find('xmin').text)
            ymin = float(bbox.find('ymin').text)
            xmax = float(bbox.find('xmax').text)
            ymax = float(bbox.find('ymax').text)

            cutting = False
            attributes = obj.find('attributes')
            if attributes is not None:
                for attr in attributes.findall('attribute'):
                    if attr.find('name').text == 'Cutting':
                        cutting = attr.find('value').text.lower() == 'true'

            objects.append({
                'label': name,
                'bbox': [xmin, ymin, xmax, ymax],
                'cutting': cutting
            })
        return objects
```

✅ 3. Load Dataset (Replace Path Accordingly)

```python
dataset_root = '/content/distribution/Train'  # adjust path as needed

dataset = CutInSequenceDataset(
    root_dir=dataset_root,
    sequence_length=5
)

print(f"Total Sequences: {len(dataset)}")
```

✅ 4. Preview Sample With Annotations

```python
def show_sample(images, annotations, index=0):
    img = images[index].permute(1, 2, 0).numpy()
    ann = annotations[index]

    fig, ax = plt.subplots(1)
    ax.imshow(img)

    for obj in ann:
        x1, y1, x2, y2 = obj['bbox']
        cutting = obj['cutting']
        label = obj['label']
        color = 'red' if cutting else 'green'
        rect = patches.Rectangle((x1 / 1920 * 512, y1 / 1080 * 512),
                                 (x2 - x1) / 1920 * 512,
                                 (y2 - y1) / 1080 * 512,
                                 linewidth=2, edgecolor=color, facecolor='none')
        ax.add_patch(rect)
        ax.text(x1 / 1920 * 512, y1 / 1080 * 512 - 10, f"{label} | {'Cut' if cutting else
'NoCut'}",
                color=color, fontsize=10, bbox=dict(facecolor='white', alpha=0.5))

    plt.axis('off')
    plt.show()
```

✅ 5. Fetch and Show a Batch

```python
images, annotations = dataset[0]
print(f"Sequence shape: {images.shape}")  # (sequence_len, 3, 512, 512)

# View a specific frame (0 to sequence_length-1)
show_sample(images, annotations, index=0)
```

Let me know if you want to:

- Add Deep SORT tracking or transform annotations
- Wrap this in a PyTorch `DataLoader`
- Export YOLO format or integrate with model training directly

Ready to proceed to next step?