Combining Datasets for Multi-task Learning (MTL)

#1 Understand the task types:

Classification: Predicts a label for the whole image (e.g., "spondylolisthesis" or "normal").
Object Detection: Finds and classifies objects in the image, giving bounding boxes (e.g., "vertebra anomaly" at [x, y, w, h]).
Segmentation: Labels each pixel in the image (e.g., highlights the exact area of a lesion).

 # 2. Dataset Preparation

Format Consistency:

- Convert all datasets to a common format (e.g., COCO, Pascal VOC, or a custom format).
- Ensure each image includes *all* available annotations: class labels, bounding boxes, and segmentation masks.

Merging Datasets:

- Combine CSVs or annotation files, ensuring no filename conflicts. Create different columns for each feature.
- If some images only have one type of annotation, you can still use them for that task.

 # 3. Unified DataLoader:

- Custom Dataset Class:
  - Write a PyTorch `Dataset` that loads:
    - The image
    - The classification label (if available)
    - The bounding boxes (if available)
    - The segmentation mask (if available)
  - Return all three (or `None` if missing) for each sample.

```
class MultiTaskDataset(Dataset):
    def __init__(self, img_paths, class_labels, bboxes, masks, transform=None):
        self.img_paths = img_paths
        self.class_labels = class_labels
        self.bboxes = bboxes
        self.masks = masks
        self.transform = transform

    def __getitem__(self, idx):
        image = Image.open(self.img_paths[idx]).convert("RGB")
        label = self.class_labels[idx] if self.class_labels else None
        bbox = self.bboxes[idx] if self.bboxes else None
        mask = self.masks[idx] if self.masks else None
        if self.transform:
```

```
        image = self.transform(image)
    return image, label, bbox, mask
```

4. Model Architecture:

Shared Backbone:
  - Use a Convolutional Neural Network (CNN) to extract features from the image.
- Task-specific Heads:
  - Classification Head: Fully connected layers for class prediction.
  - Detection Head: Layers for bounding box regression and class prediction.
  - Segmentation Head: Upsampling layers for pixel-wise mask prediction.

```
class MultiTaskModel(nn.Module):
    def __init__(self, backbone):
        super().__init__()
        self.backbone = backbone
        self.class_head = nn.Linear(..., num_classes)
        self.det_head = DetectionHead(...)  # e.g., YOLO or Faster R-CNN style
        self.seg_head = SegmentationHead(...)  # e.g., UNet style

    def forward(self, x):
        features = self.backbone(x)
        class_out = self.class_head(features)
        det_out = self.det_head(features)
        seg_out = self.seg_head(features)
        return class_out, det_out, seg_out
```

5. Loss Functions:

- Classification: CrossEntropyLoss or BCEWithLogitsLoss
- Detection: Combination of classification loss and bounding box regression loss (e.g., SmoothL1Loss)
- Segmentation: Dice loss, BCE, or CrossEntropy for masks

Combine losses (with weights if needed):

```
total_loss = cls_loss + alpha * det_loss + beta * seg_loss
```

6. Training Loop:

- For each batch:
  - Forward pass through the model with a dedicated number of epochs
  - Compute each loss (skip if annotation is missing)
  - Backpropagate the total loss
  - Track metrics for each task

 # 7. Evaluation:

- Evaluate each task separately using appropriate metrics:
  - Classification: Accuracy, ROC-AUC
  - Detection: mAP (mean Average Precision)
  - Segmentation: IoU, Dice score

 # 8. Tips for Combining Datasets:

- Data Augmentation: Use the same augmentations for all tasks.
- Missing Annotations: If an image lacks some annotations, only compute the loss for available tasks.
- Balancing: If datasets are imbalanced, consider oversampling or weighted losses.

 # 9. Example Workflow:

1. **Convert all datasets** to a unified format (e.g., COCO).
2. **Write a custom Dataset** class to load all annotation types.
3. **Build a multi-head model** with a shared backbone.
4. **Train** using combined losses.
5. **Evaluate** each task separately.

Summary:

By merging datasets and using a multi-task model, we can train on classification, detection, and segmentation together. This approach leverages *all* available data and can improve performance by sharing learned features across tasks.