Tumor Detection & Segmentation

Deliverables:

- Data preparation process and brief description of the models and techniques used in each task.
- Training and valid performance metrics for all three models (object detection, segmentation on full image and segmentation on cropped detection).
- Screenshots of the validation set results in both phases (detection and segmentation)

Tumor Segmentation:

In this section, binary segmentation was performed on grayscale medical images to detect tumors. The goal was to segment the entire tumor area in each image. The base model used for this task was **U-Net**, which was implemented from scratch, including its double convolution layers, contracting path (encoder), and expanding path (decoder).

Each input image was accompanied by a corresponding ground truth mask. These masks are binary: **white pixels indicate tumor regions**, and **black pixels represent the background**—a typical setup for **semantic segmentation** (not instance segmentation, which would require distinguishing between multiple tumor instances if they existed).

All images and masks were preprocessed for optimal training conditions:

- Converted to grayscale (if not already)
- Resized to **256 × 256 pixels**
- Transformed into tensors

We experimented with **two training approaches** to explore how different data splits impact model performance.

Approaches:

First Approach:

The dataset was split into training and validation subsets using a fixed random seed (`random_state`) to ensure reproducibility and prevent data leakage. The provided validation set was reserved for final testing.
To preserve model progress and facilitate early stopping, **checkpoints were saved periodically** during training.

Second Approach:

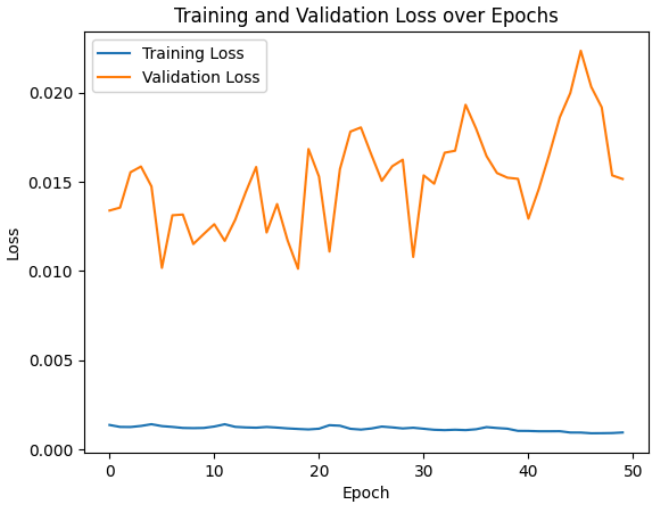
All available training data was used to train the model, while the provided validation set was used directly for validation. Preprocessing steps were identical to those in the first approach. Like before, model checkpoints were saved after specific epochs.

Results:

First Approach:
Achieved:

- Training Loss: **0.0010**
- Validation Loss: **0.0152**
- Test Loss: **0.0024**

```
Epoch [43/50] Started
Epoch [43/50] Finished, Train Loss: 0.0010, Val Loss: 0.0165

Epoch [44/50] Started
Epoch [44/50] Finished, Train Loss: 0.0010, Val Loss: 0.0186

Epoch [45/50] Started
Epoch [45/50] Finished, Train Loss: 0.0010, Val Loss: 0.0200

Epoch [46/50] Started
Epoch [46/50] Finished, Train Loss: 0.0009, Val Loss: 0.0223

Epoch [47/50] Started
Epoch [47/50] Finished, Train Loss: 0.0009, Val Loss: 0.0203

Epoch [48/50] Started
Epoch [48/50] Finished, Train Loss: 0.0009, Val Loss: 0.0192

Epoch [49/50] Started
Epoch [49/50] Finished, Train Loss: 0.0009, Val Loss: 0.0154

Epoch [50/50] Started
Epoch [50/50] Finished, Train Loss: 0.0010, Val Loss: 0.0152
```



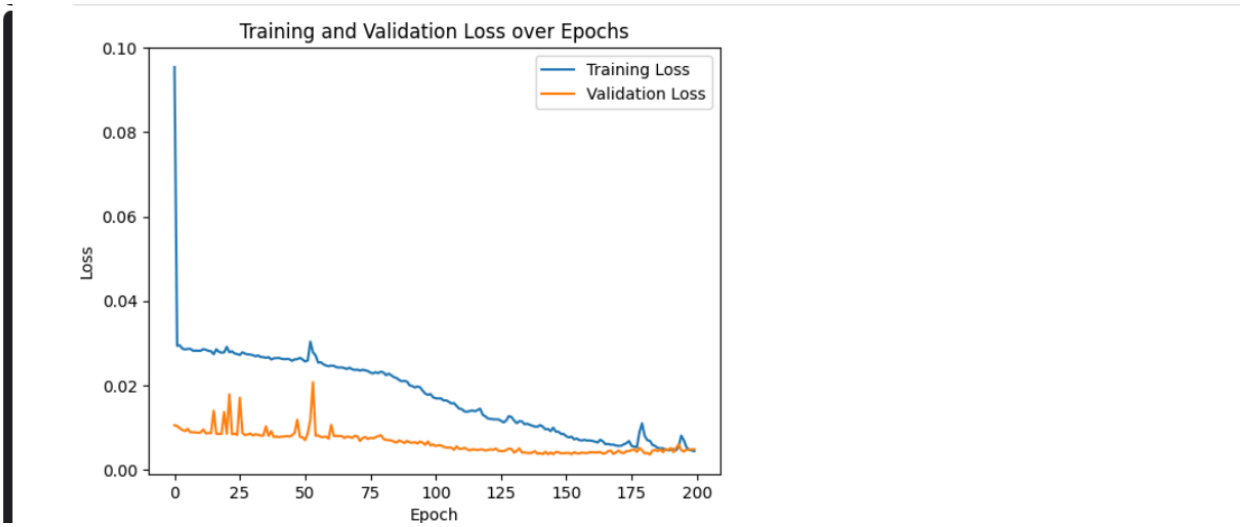

```
[26] # Evaluate on test data
     test_loss = evaluate_model(test_dataloader, model, criterion, device)
```

```
Test Loss: 0.0024
```

Second Approach:
Achieved:

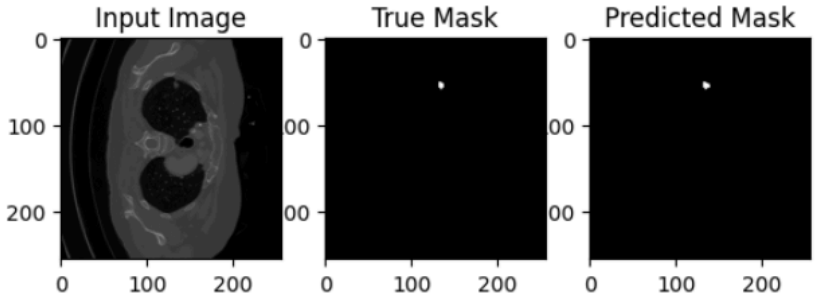
- Training Loss: **0.0044**
- Validation Loss: **0.0048**

```
Epoch [195/200] Started
Epoch [195/200] Finished, Train Loss: 0.0080, Val Loss: 0.0046

Epoch [196/200] Started
Epoch [196/200] Finished, Train Loss: 0.0069, Val Loss: 0.0043

Epoch [197/200] Started
Epoch [197/200] Finished, Train Loss: 0.0053, Val Loss: 0.0047

Epoch [198/200] Started
Epoch [198/200] Finished, Train Loss: 0.0047, Val Loss: 0.0047

Epoch [199/200] Started
Epoch [199/200] Finished, Train Loss: 0.0045, Val Loss: 0.0048

Epoch [200/200] Started
Epoch [200/200] Finished, Train Loss: 0.0044, Val Loss: 0.0048
```



These results indicate possible **underfitting**, suggesting the model struggled to learn from the full training set.

Inference:



Areas for Improvement:

- **Train for more epochs** to allow the model to better fit the data.
- **Increase dataset size** through:
    - **Data augmentation** (e.g., flipping, rotation, elastic deformation) to introduce more variability and reduce overfitting.
    - **Collecting additional annotated samples** to directly address underfitting.
- Experiment with **alternative architectures** such as **Mask R-CNN** or **DeepLabv3+** for potentially better performance on complex segmentation tasks.

Tumor Detection:

In this section, object detection was performed on medical images to localize and classify lung tumors. Detection outputs bounding boxes around tumors along with their confidence scores. The primary goal was to accurately identify all tumor instances within each image.

We experimented with **two training approaches.**

Approaches:

- **Two Stage detector** which is Faster-R-CNN model to perform object detection as it tries to use Uses a Region Proposal Network (RPN) to generate candidate regions followed by ROI-based classification
- **Single Stage Detector** which is a YOLO-v8 model as it predicts bounding boxes and class probabilities in a single pass.
  Processes images in real-time, ideal for scalable deployments.

Tumor Detection (Faster-R-CNN):

Preprocessing:-

- Loaded and unzipped the dataset containing lung images and annotations.
- Implemented **TumorDataset** class to handle image loading, annotation parsing, and data augmentation.
- Validated dataset integrity with test cases for annotation loading and bounding box transformations.

Model Architecture:-

- **Backbone**: Used pretrained VGG16 (frozen early layers) for feature extraction.
- **RPN (Region Proposal Network)**: Generated candidate regions with multi-scale anchors.
- **ROI Head**: Classified regions and refined bounding box coordinates.

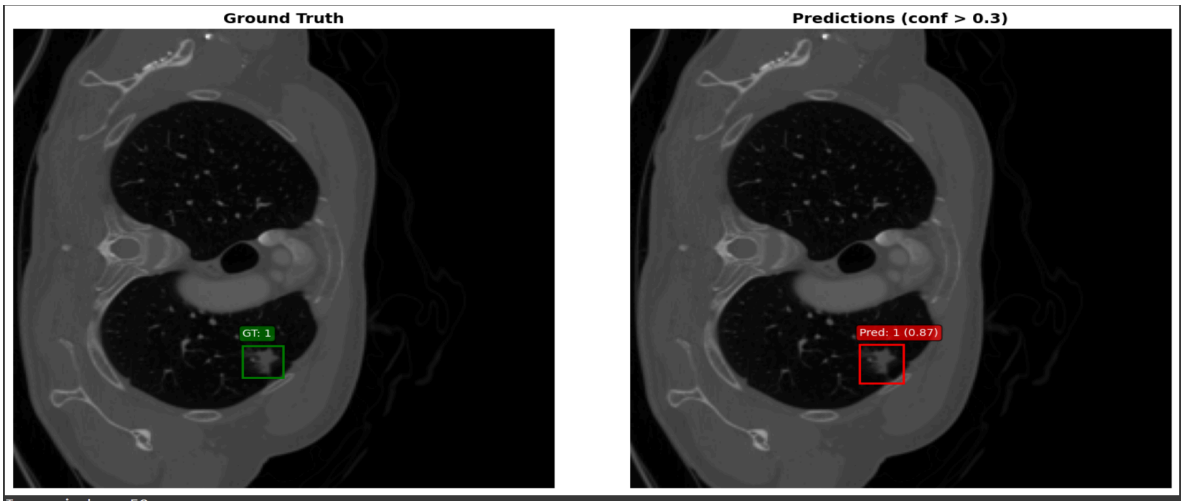
Training:-

- Defined hyperparameters (learning rate, batch size, epochs) in a YAML config file.
- Tracked losses (RPN classification/localization, ROI classification/localization).
- Saved checkpoints periodically.
- Model converged with decreasing losses over epochs.
- Final model saved as `final_faster_rcnn_lung_tumor.pth`.

```
Finished epoch 15
Epoch 15/15 - RPN Cls: 0.0113 | RPN Loc: 0.8130 | ROI Cls: 0.1161 | ROI Loc: 0.0074 | Total: 0.9477
Done Training...
```

Inference:-

- Evaluated on a seperate test set using average loss
- Visualized predictions vs. ground truth for sample image

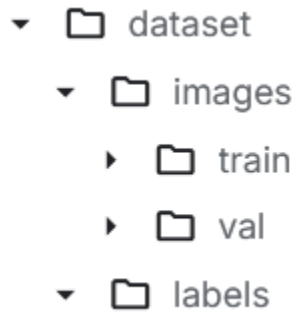



```
Image index: 50
Ground Truth: 1 boxes
Predictions: 1 boxes (confidence > 0.3)
GT Labels: [1]
Pred Labels: [1]
Pred Scores: [0.873213]
```

Tumor Detection (Pretrained YOLO model):

Preprocessing:-

- Convert Directories of Tasks to Images and Labels



- Read the original annotation file(s), parse bounding box coordinates and convert them to YOLO format **(class_id center_x center_y width height).**
- Normalize each value by image width and height.
- If an image has **no tumor**, create an **empty .txt label file** (same name as the image) to ensure YOLO format consistency.
- Create a **data.yaml** file with the following format

```
train: /content/dataset/images/train
val: /content/dataset/images/val
nc: 1
names: ['tumor']
```

Training:-

- Use the Ultralytics YOLOv8 model for lung tumor detection training.
- First Training Attempt :-

  - **Image size**: 640.
  - **Number of epochs**: 50.

- Second Training Attempt (Tuned Hyperparameters):

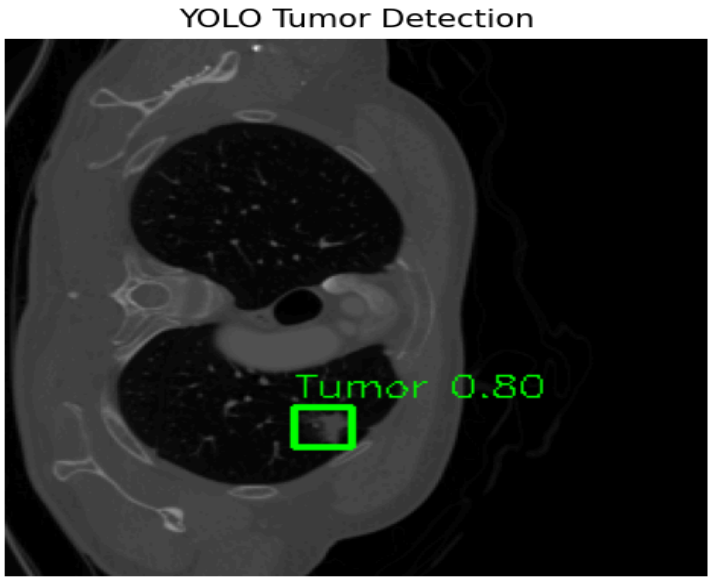
- **Epochs**: 50.
- **Image size**: 1024.
- **Patience**: 10 (early stopping).
- **Initial learning rate**: 1e-4.
- **Close mosaic augmentation**: after epoch 10.
- **Box loss gain**: 0.05.
- **Class loss gain**: 0.5.

Training results:-

```
fitness: 0.4191974262076161
keys: ['metrics/precision(B)', 'metrics/recall(B)', 'metrics/mAP50(B)', 'metrics/mAP50-95(B)']
maps: array([     0.3892])
names: {0: 'tumor'}
plot: True
results_dict: {'metrics/precision(B)': 0.8106659296248288, 'metrics/recall(B)': 0.6395348837209303, 'metrics/mAP50
(B)': 0.689138561705946, 'metrics/mAP50-95(B)': 0.38920396670780166, 'fitness': 0.4191974262076161}
save_dir: PosixPath('runs/detect/train')
speed: {'preprocess': 0.46640094898270484, 'inference': 6.717970948979461, 'loss': 0.0006327551016845379, 'postproce
ss': 4.050487010206371}
task: 'detect'
```

Inference:

- Load the trained YOLOv8 detection model.
- Apply tumor detection to a single image and draw bounding boxes around detected tumors.
- Visualize the detection results.

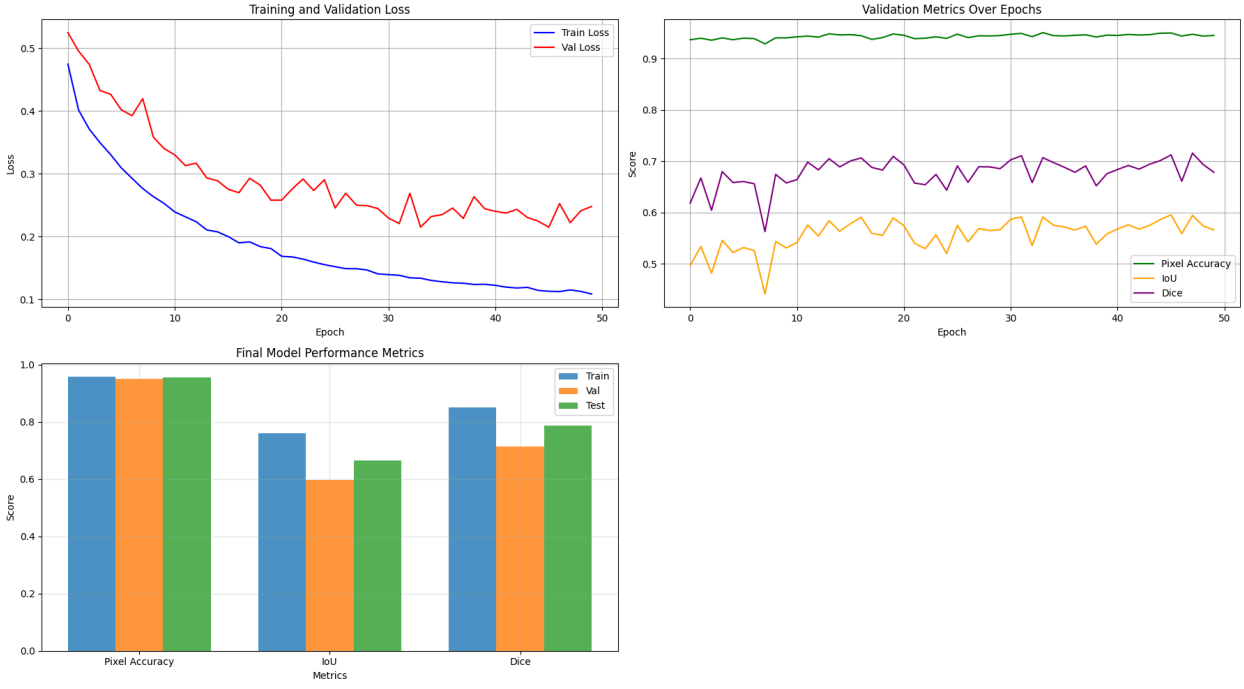


Segmentation of detected tumors:

The goal of this approach is to segment tumors after detection, by segmenting the cropped version of the medical image using coordinates from detection models. U-Net was used to segment RGB medical images. As a preprocessing step **`crop_and_save`** function extracts and saves tumor regions from medical images based on provided bounding boxes. It handles both training and inference scenarios, with additional padding applied during inference to retain surrounding context. For very small tumors, it expands the crop to a minimum size (32×32) to ensure adequate spatial context. Each cropped image and its corresponding mask are resized to 64×64 pixels and saved. The function also records metadata, including crop dimensions, which can be used for further analysis or tracking during the training pipeline.

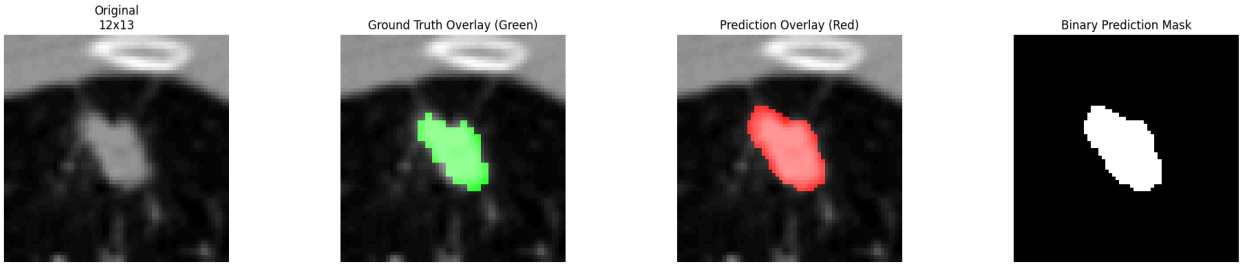
Training:

- Training Loss: **0.1127**
- Validation Loss: **0.2146**
- Test accuracy: **0.9501**



Inference:

Cropped image



Mapped back to original image