# Detection of Liver Tumors

AI For Healthcare - Ex2

Using Architectures DETR & YOLOv7

Shir Nitzan, Timor Baruch, Hadar Pur

June 17, 2023

## 1 Overview

In this exercise, we focused on the detection of liver tumors using contrast-enhanced CT images. The dataset comprised of 201 images from patients suffering from primary cancers and metastatic liver disease, often a consequence of colorectal, breast, and lung primary cancers. The data, obtained from the IRCAD Hôpitaux Universitaires, Strasbourg, France, is a subset of patients from the 2017 Liver Tumor Segmentation (LiTS) challenge [1]. The dataset is accessible at the Medical Decathlon website[1].

The primary challenge was the significant label imbalance between large (liver) and small (tumor) target Regions of Interest (ROIs). To mitigate this, we implemented a balanced sampling strategy, selecting an equal number of images from each class. Our approach involved a comprehensive exploration of the data, rigorous pre-processing, and the application of the YOLO architecture for object detection. We also experimented with various data augmentation techniques and hyperparameter settings to optimize our model's performance.

## 2 Data Exploration

The data exploration phase was a critical step in our project, as it allowed us to understand the distribution of the data, visualize the images, and evaluate the initial results. One of the key challenges we faced was the significant imbalance in the data. The dataset was skewed, with the majority of the volumes not containing any of the classes - neither liver nor tumor. This imbalance was even more pronounced when it came to the slices containing tumors, which were few and far between.

This imbalance posed a significant challenge for the network's ability to detect tumors. The scarcity of tumor-containing slices in the dataset could lead the network to develop a bias towards predicting the absence of tumors, thereby reducing its sensitivity to cases where tumors are present.

To address this issue, we implemented a balanced sampling strategy. We selected an equal number of images from each class, resulting in a total of 21,570 images - 7,190 images with tumors, 7,190 images with livers,

[1] Medical Segmentation Decathlon

and 7,190 images with none of them. This approach helped to ensure a more balanced representation of the classes in our training data, thereby improving the network's ability to detect tumors.

# 3 Data Labeling and Conversion

Our dataset initially consisted of 3D CT images and segmentations. However, our assignment required us to perform object detection on this data, which necessitated the creation of labels in the form of bounding boxes for the liver and tumors.

To create these labels, we computed the bounding boxes around the liver and tumors in each 3D image. This process involved identifying the minimum and maximum x and y coordinates that encapsulate the liver and tumors, which defined the boundaries of our bounding boxes.

In addition to creating labels, we also needed to convert our 3D images into 2D slices. This conversion was necessary because the YOLO architecture, which we used for our model, operates on 2D images.

Finally, we had to convert the format of our images and labels into a format that the YOLO architecture could accept. This process involved saving the 2D slices as individual image files and creating a corresponding text file for each image, containing the labels for that image in the format required by YOLO.

# 4 Data Preprocessing

## 4.1 Data Normalizations

We employed three types of normalizations in our data preprocessing stage: Z-score, Min-Max, and Percentile Normalization.

- **Z-score Normalization:** This method standardizes the features by subtracting the mean and dividing by the standard deviation. It is particularly useful when the data follows a Gaussian distribution.
- **Min-Max Normalization:** This method rescales the features to a fixed range, usually 0 to 1. It is beneficial when the dataset contains mixed types of attributes (i.e., binary, categorical, and continuous).
- **Percentile Normalization:** This method scales the data according to a specified percentile range. We used the 1st and 99th percentiles, which significantly enhanced the contrast and created a good distribution of the data. This was particularly beneficial for our dataset as it improved the visibility of the tumors.

After experimenting with these normalization methods, we found that Percentile Normalization was the most effective for our dataset. It provided the best contrast and distribution, which in turn improved the model's ability to detect tumors in the CT images.

To illustrate the effect of the Percentile Normalization, we present below the images before and after the normalization process.

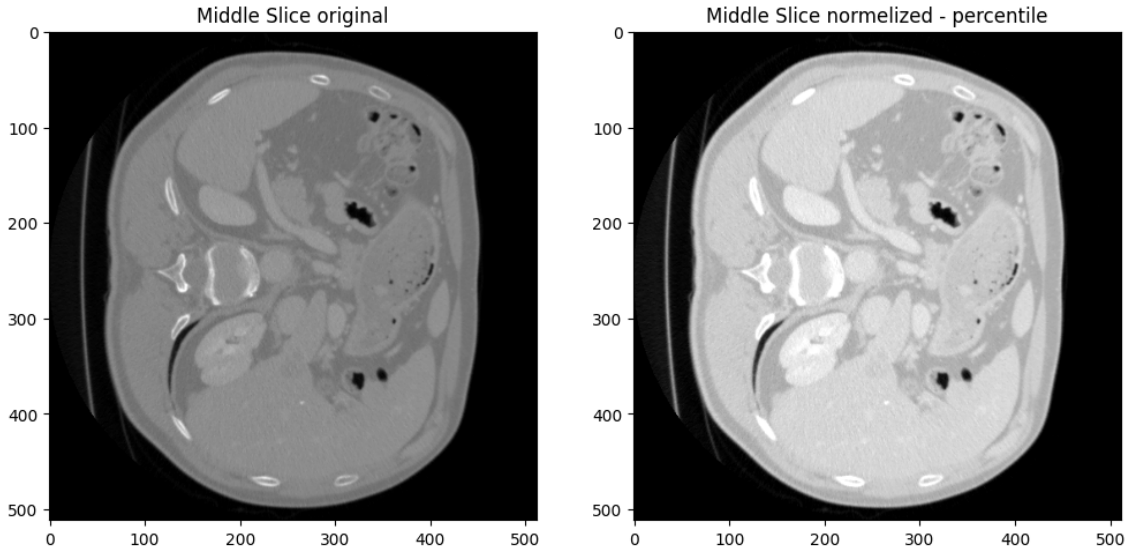



Figure 1: Comparison of images before (left) and after (right) Percentile Normalization. The normalization process enhances the contrast and improves the visibility of the tumors.

## 4.2 Data Augmentation

We implemented several data augmentation techniques to enhance the diversity of our dataset and improve the generalization capabilities of our model. The original image was subjected to a series of augmentation techniques that were applied sequentially, with a certain probability, modifying the image with each step to create the augmented version. These techniques included flipping, rotating, translating, and scaling the images.

- **Flip:** This method creates a mirrored version of the image. It helps to increase the size of the dataset and reduce overfitting by providing variations of the same image.
- **Rotate:** This method rotates the image by a certain angle. It is useful for creating a more robust model that can recognize the object in different orientations.
- **Translate:** This method shifts the image along the x or y direction. It helps the model to recognize the object regardless of its position in the image.
- **Scaling:** This method resizes the image. It helps the model to recognize the object regardless of its size in the image.

These augmentation techniques were beneficial in creating a more diverse dataset, which in turn improved the model's ability to generalize to unseen data.

To illustrate the effect of the data augmentation, we present below the images before and after the augmentation process.

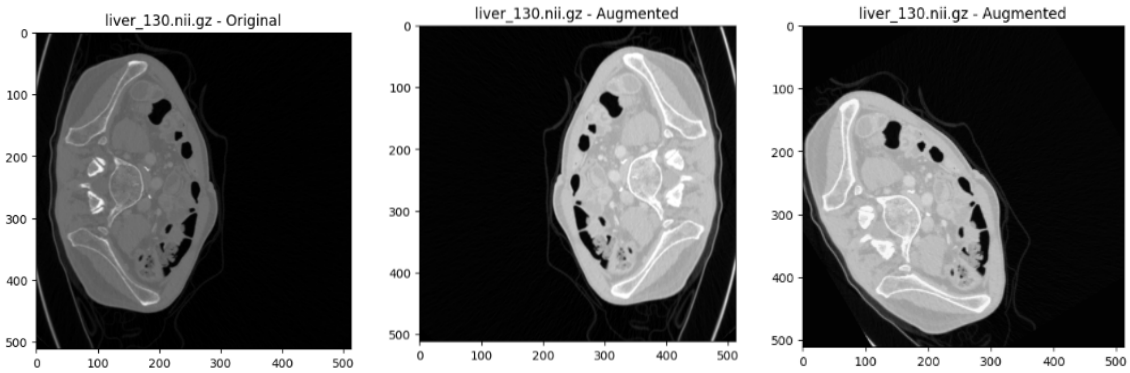



Figure 2: Comparison of images before (left) and after (right) data normalization and augmentation. The augmentation process enhances the diversity of the dataset and improves the model's ability to generalize.

# 5 Model Architecture

We utilized the YOLO (You Only Look Once) architecture for our model [2] after experiencing with DETR architecture [3].

## 5.1 DETR

The DETR (Detection Transformer) architecture is a state-of-the-art object detection framework that combines Transformers and convolutional neural networks (CNNs) for accurate object detection in medical images.

In general, DETR uses a CNN backbone to extract features from the input image. These features are then passed through a Transformer encoder, which captures both local and global contextual information. Object queries, representing the objects to be detected, attend over the encoder output to predict object presence, class, and location.

## 5.2 YOLO

YOLO (You Only Look Once) is an efficient object detection algorithm renowned for its real-time capabilities. It takes a unique approach by treating object detection as a single regression problem. The input image is divided into a grid, and YOLO predicts bounding boxes and class probabilities for objects within each grid cell. This grid-based methodology enables YOLO to process the entire image simultaneously, resulting in impressive speed.

YOLO utilizes a convolutional neural network (CNN) backbone, such as Darknet or ResNet, to extract informative features from the input image. These features are then used to predict bounding box coordinates, object class probabilities, and confidence scores. To handle variations in object sizes and shapes, YOLO incorporates anchor boxes.

One of YOLO's notable strengths is its ability to detect multiple objects within a single grid cell, effectively reducing redundancy and overlapping detections. This efficient design makes it well-suited for scenarios involving small objects and intricate details, such as autonomous driving and surveillance applications.

### 5.3 From DETR to YOLO

Our primary objective was to utilize the DETR model for liver tumor detection. However, despite our efforts, we encountered challenges in accomplishing this mission effectively. Several factors may have contributed to the suboptimal performance of DETR in this context.

Limited success of the DETR model in liver tumor detection could be attributed to several factors. Firstly, the scarcity or imbalance of liver tumor samples in the training dataset may have hindered the model's ability to learn and generalize effectively. Insufficient instances, small tumors detected or an unbalanced distribution of tumor variations could have impeded accurate detection in unseen cases.

Secondly, the complex nature of medical imaging data, including variations in image quality, tumor appearance, and the presence of artifacts, might have posed challenges for DETR's architecture. The model may have lacked the necessary capacity to capture the diverse range of tumor characteristics adequately.

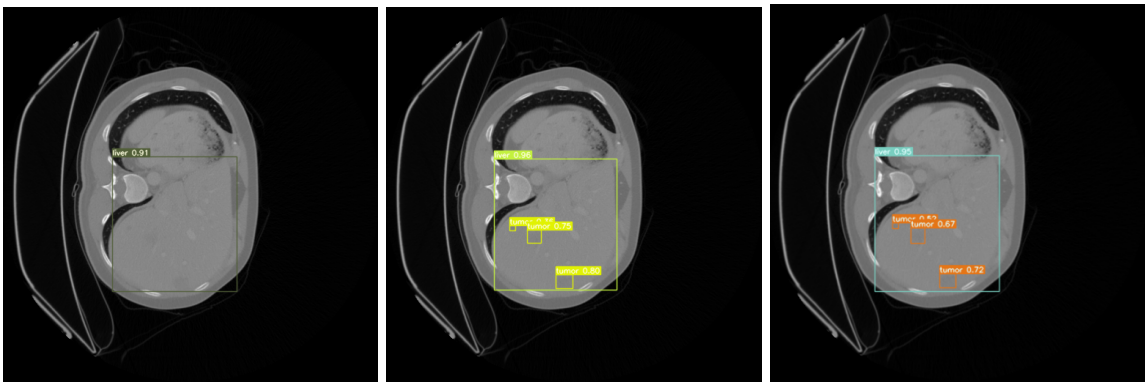
Moreover, the preprocessing steps and data augmentation applied, such as converting images from 2D to 3D and transforming data into the COCO format, might have introduced challenges or inconsistencies affecting the model's performance in detecting liver tumors accurately.

After initially adopting the DETR architecture for liver tumor detection, we encountered unsatisfactory results and observed that the model struggled to learn effectively. As a result, we made the decision to transition to the YOLOv7 architecture, which is the best YOLO version in terms of accuracy (MAP), which yielded superior performance. The single-stage detection approach employed by YOLO, along with its capability to handle small objects and intricate details, proved to be advantageous for the complexity of our medical imaging data.

## 6 Hyperparameter Tuning

We experimented with different learning rates: 0.1, 0.01, and 0.001. The learning rate is a crucial hyperparameter that determines how much the model changes in response to the estimated error each time the model weights are updated. Choosing the right learning rate is essential for training an effective model.

After conducting experiments, we found that a learning rate of 0.01 yielded the best results. This learning rate was neither too large, which could lead to erratic behavior, nor too small, which could result in the model learning too slowly. It provided a good balance, allowing the model to learn at a steady pace while also enabling it to adjust its weights and biases effectively to minimize the loss.




lr = 0.1 lr = 0.01 lr = 0.001

Figure 3: Liver and Tumor detection with different learning rates

In addition to the learning rate, we also set the batch size to 32. The batch size determines the number of training examples used in one iteration. A batch size of 32 was chosen as it is a common choice in practice, offering a good balance between computational efficiency and model performance. It is large enough to benefit from the speedup of matrix operations in the deep learning libraries, yet small enough to not require excessive memory resources.

We also used the Adam optimizer for our model. Adam, short for Adaptive Moment Estimation, is an optimization algorithm that can handle sparse gradients on noisy problems. It's known for its efficiency and the relatively low memory requirements. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle a wide range of data and tasks.

Lastly, we used weight decay, which is a regularization technique that prevents the weights from growing too large by adding a penalty term to the loss function. This helps to prevent over-fitting by adding a cost to the loss function for large weights.

# 7 Training

Our model was trained using either half or the entire dataset for a range of 15 to 30 epochs. The dataset for liver tumor analysis comprises 3D pictures in the nii.gz format. As the test images lacked labels, they were excluded from the dataset. The remaining 131 train images were divided into three sets: 80% for training, 10% for validation, and 10% for testing purposes. Given the size of our dataset and the complexity of the task, this process took a significant amount of time. However, due to resource constraints, specifically GPU limitations, we were unable to run more epochs.

## 7.1 DETR

In order to leverage the preprocessed data in COCO format, which was initially prepared for the DETR architecture, we aimed to adapt it for YOLO. To achieve this, we developed a converter that transformed the COCO format to YOLO's accepted format which includes a text file for each slice with its labels and

bounding boxes indices and a yaml file. By creating this converter, we were able to seamlessly transition the preprocessed data from COCO to YOLO, taking advantage of the hard work we did before.

We present the following performance metrics obtained on the validation and training sets of the dataset. It is observed that the training loss exhibits a decaying trend, reaching a value of 0.741 after 10 epochs on a subset of 500 images of each type; liver, tumor, and background classes. However, the validation loss does not demonstrate stability. Notably, when conducting the experiment on a larger dataset consisting of 7000 images per type, similar results were achieved on the validation set.

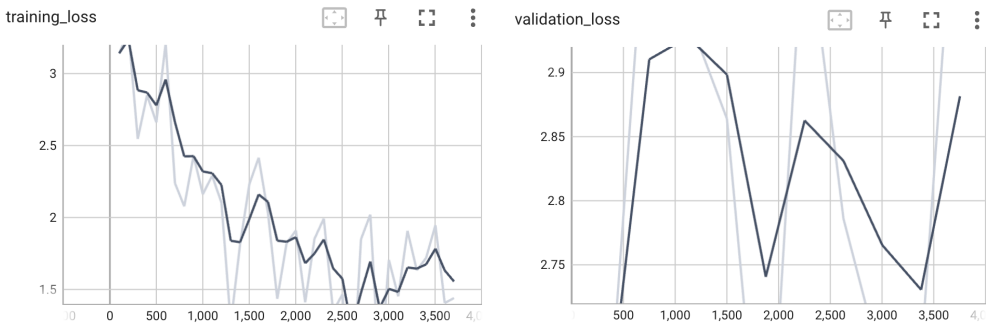



Figure 4: Validation and training loss for DETR

## 7.2 YOLO

### 7.2.1 Experiments 1 and 1.1

In both experiments, we utilized the entire dataset comprising 131 images for each set; train, test and validation. A learning rate of 0.01 and a batch size of 32 were employed. The primary distinction between these experiments lay in the number of epochs. Specifically, in experiment 1, the model underwent training for 30 epochs, whereas in experiment 1.1, the model was trained for 15 epochs.

#### 7.2.1.1 Experiment 1

In this experiment, we used all the data (131 images), a learning rate of 0.01, a batch size of 32, and trained the model for 30 epochs. Our observations indicate that the results obtained from this experiment were similar to those of experiment 1.1, which involved a reduced number of epochs also on all the dataset. This suggests that the number of epochs does not significantly affect the model's accuracy. Rather, it implies that increasing the amount of available data would be more advantageous in enhancing the model's performance, rather than solely relying on increasing the number of epochs.

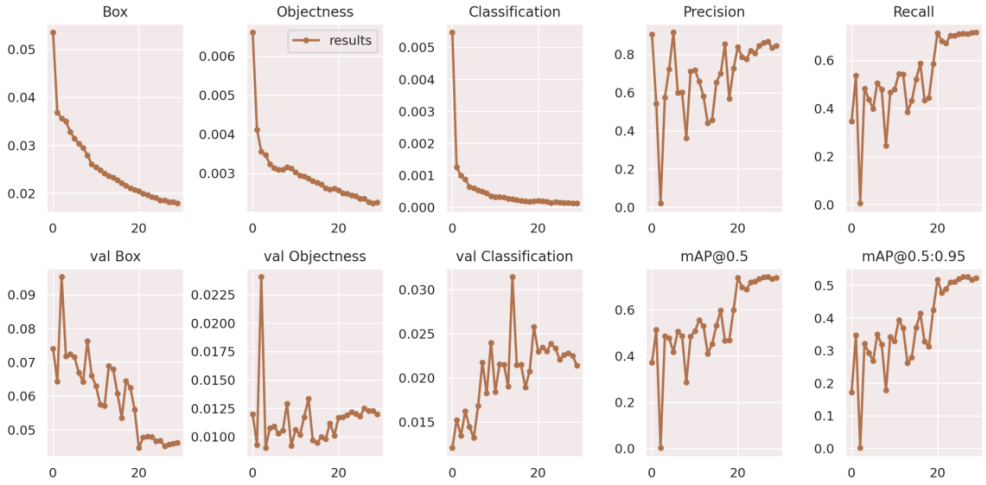



Figure 5: Training and Validation progress for Experiment 1.

#### 7.2.1.2 Experiment 1.1

In this experiment, we used all the data (131 images), a learning rate of 0.01, a batch size of 32, and trained the model for 15 epochs.

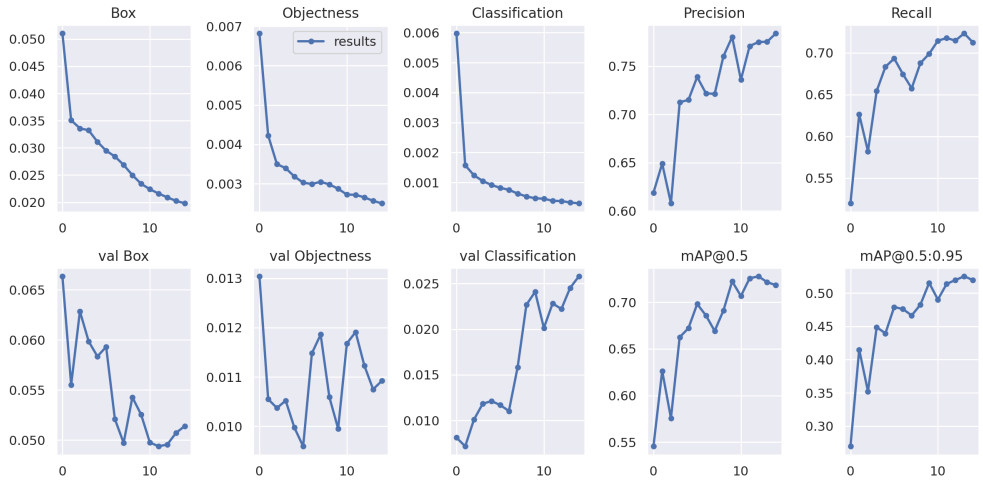



Figure 6: Training and Validation progress for Experiment 1.1.

The training progress image shows how the model improved over time. As the number of epochs increased, the model's performance on the training data improved.

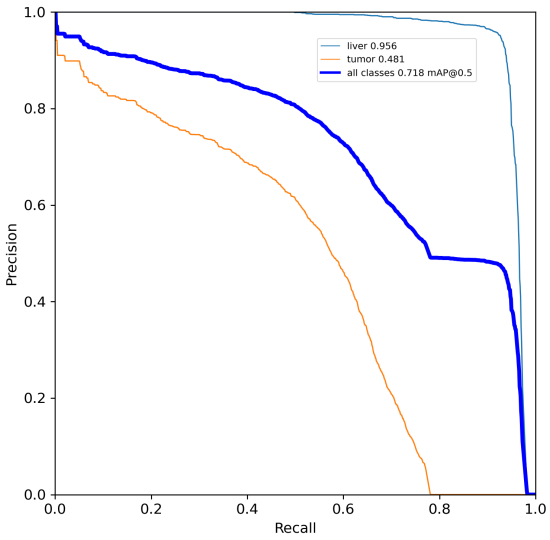



Figure 7: Precision-Recall curve for Experiment 1.1.

The Precision-Recall curve shows the trade-off between precision and recall for different threshold. A high area under the curve represents both high recall and high precision.

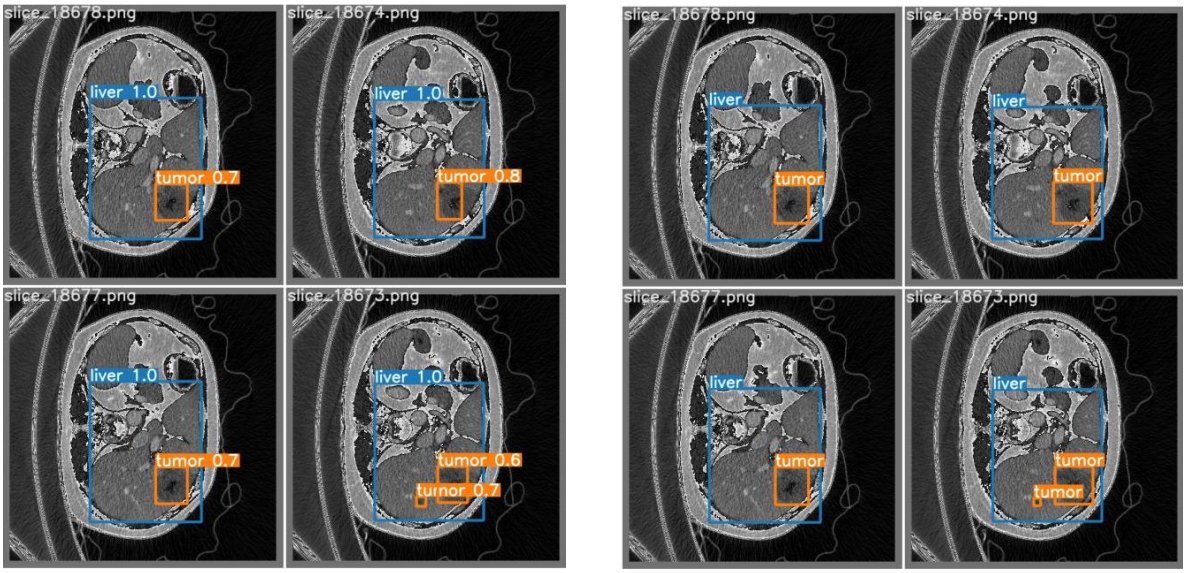



Figure 8: Comparison of predictions (left) and ground truth (right) for Experiment 1.1.

The left image shows the predictions made by the model, and the right image shows the ground truth. The bounding boxes in the images represent the locations of the objects detected by the model.

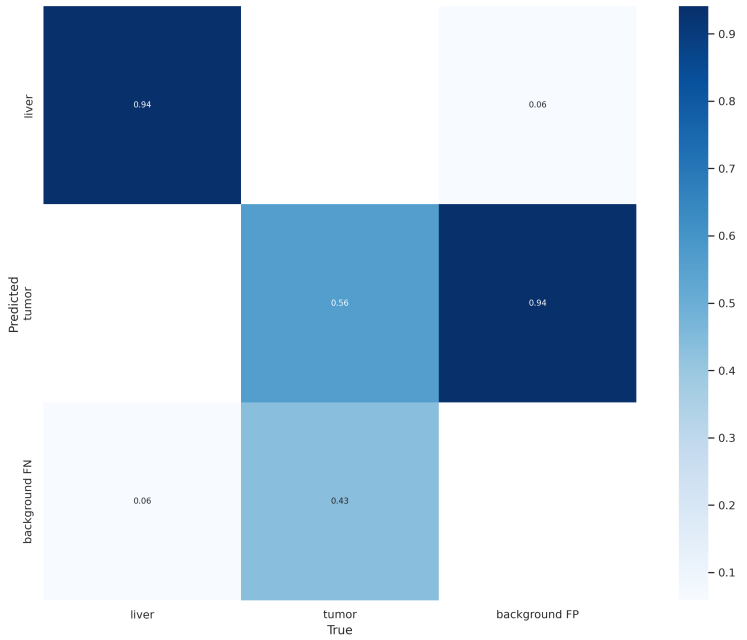



Figure 9: Confusion matrix for Experiment 1.1.

The confusion matrix provides a more detailed view of the model's performance. It shows the number of true positive, true negative, false positive, and false negative predictions.

In this experiment, the model performed well on the training data, but there was room for improvement on the test data. This suggests that the model might be over-fitting to the training data, and adjustments to the model's complexity or the amount of training data might be necessary.

The model's performance improved over time, with the precision, recall, and F1 score all increasing over the epochs. However, the model's performance on the validation data did not improve as much, suggesting that the model might be over-fitting to the training data.

The results of this experiment suggest that the model's architecture and training parameters were generally effective, but there may be room for improvement. The learning rate of 0.01 and batch size of 32 were suitable for this experiment. They enabled the model to learn steadily without excessive loss fluctuations and ensured effective weight updates during training without instability. Future experiments could explore using different model architectures, training parameters, or data augmentation techniques to further improve the model's performance.

### 7.2.2 Experiment 2, 2.1 and 2.2

In all three experiments, we made use of the complete dataset consisting of 65 images. A batch size of 32 was employed for each experiment, and the number of epochs was set to 15. The main differentiating factor among these experiments was the learning rate. In experiment 2, a learning rate of 0.01 was utilized. In experiment 2.1, the learning rate was set to 0.001, and in experiment 2.2, it was increased to 0.1.

#### 7.2.2.1 Experiment 2

In Experiment 2, we used half of the data (65 images), a learning rate of 0.01, a batch size of 32, and trained the model for 15 epochs. The model's performance improved over time, with the precision, recall, and F1 score all increasing over the epochs. However, the model's performance on the validation data did not improve as much, suggesting that the model might be overfitting to the training data. The learning rate of 0.01 seemed to be a good choice for this experiment, as it allowed the model to learn at a steady pace without causing the loss to fluctuate too much. The batch size of 32 also seemed to be a good choice, as it allowed the model to update its weights frequently enough to learn effectively, but not so frequently that the training process became unstable.

#### 7.2.2.2 Experiment 2.1

Experiment 2.1 was similar to Experiment 2, but with a lower learning rate of 0.001. This lower learning rate resulted in a slower but more stable learning process. However, despite the increased stability, the lower learning rate did not yield a significant improvement in the model's performance. This suggests that while a lower learning rate can lead to more stable learning, it may not always lead to better performance.

#### 7.2.2.3 Experiment 2.2

In Experiment 2.2, we increased the learning rate to 0.1. This higher learning rate resulted in a faster learning process. However, the model's performance on the validation data did not improve as much as in the previous experiments, suggesting that the higher learning rate may have caused the model to converge to a less optimal solution.

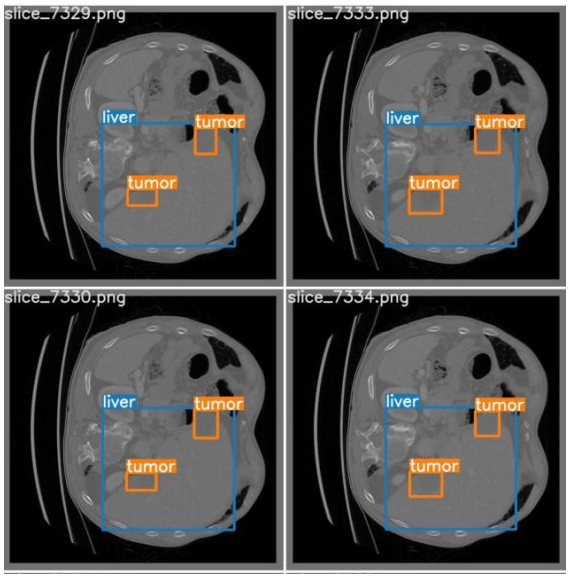
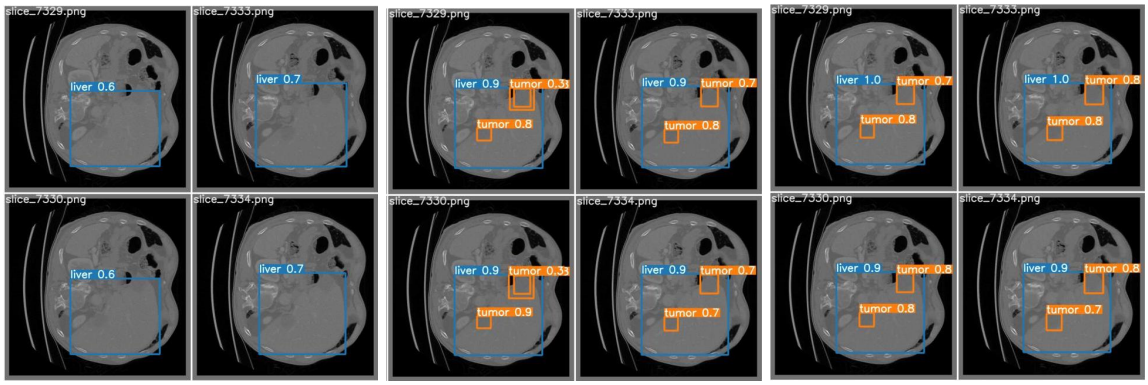



Figure 10: The ground truth for Experiment 2.




lr = 0.1 lr = 0.01 lr = 0.001

Figure 11: The predictions for three different learning rates for Experiment 2, 2.1 and 2.2

In summary, these experiments suggest that a learning rate of 0.01 provides the best balance between learning speed and model performance for our specific dataset and model architecture. While both lower and higher learning rates were able to facilitate learning, they did not yield better performance. Future experiments could explore using different learning rates to further optimize the model's performance.

#### 7.2.3 Experiment 3

In Experiment 3, we used half of the data (65 images), applied data augmentation techniques (rotation, flipping, and translation), set the learning rate to 0.01, batch size to 32, and trained the model for 15 epochs. The data augmentation techniques were used to artificially increase the size of the dataset and introduce more variability, which can help improve the model's ability to generalize to unseen data.

#### 7.2.4 Experiment 4

In Experiment 4, we focused on the impact of intensity normalization on our model's performance. We used half of the dataset (65 images), a learning rate of 0.01, a batch size of 32, and trained the model for 15 epochs.

We tested various normalization techniques, including Z-score, Min-Max, and Percentile normalization. Our focus was on the percentile intensity normalization, as it provided the best contrast, making the tumor and liver more distinguishable.

The results showed a consistent decrease in loss across epochs, and the precision, recall, and F1 score showed an upward trend. However, it's important to note that while the percentile intensity normalization enhanced the visibility of the tumor and liver in the images, it's not clear-cut whether this preprocessing step led to an improvement in the model's performance.

#### 7.2.5 Experiment 5

In Experiment 5, we combined the techniques from the previous experiments to see how they would work together. We used half of the dataset (65 images), a learning rate of 0.01, a batch size of 32, and trained the model for 15 epochs.

In this experiment, we applied both data augmentation and intensity normalization to the images. The data augmentation techniques included rotation, flipping, and translation, which were intended to increase the diversity of the training data and help the model generalize better. The intensity normalization, specifically percentile intensity normalization, was used to enhance the contrast in the images, making the tumor and liver more distinguishable.

The combination of these techniques aimed to provide the model with a more varied and representative training set, while also enhancing the quality of the images for better feature extraction. The results of this experiment would provide insights into how these preprocessing steps work in tandem and their combined effect on the model's performance.

### 7.3 Comparison of Experiments

We conducted several experiments to evaluate the performance of our model on different classes: all, liver, and tumor. The key metrics we focused on were Precision (P), Recall (R), and Mean Average Precision at different thresholds (mAP@.5, mAP@.5:.95). The results are presented separately for the training and testing phases.

| Exp | Info | Data | Eps | Cls | Img | Lbl | P | R | mAP@.5 | mAP@.5:.95 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Original | All | 30 | All | 2157 | 4188 | 0.846 | 0.716 | 0.739 | 0.522 |
| 1 | Original | All | 30 | Liv | 2157 | 2157 | 0.942 | 0.945 | 0.967 | 0.809 |
| 1 | Original | All | 30 | Tum | 2157 | 2031 | 0.749 | 0.487 | 0.511 | 0.253 |
| 1.1 | Original | All | 15 | All | 2157 | 6019 | 0.784 | 0.713 | 0.718 | 0.52 |
| 1.1 | Original | All | 15 | Liv | 2157 | 2157 | 0.952 | 0.926 | 0.956 | 0.834 |
| 1.1 | Original | All | 15 | Tum | 2157 | 3862 | 0.616 | 0.499 | 0.481 | 0.205 |
| 2 | Original | Half | 15 | All | 838 | 2754 | 0.849 | 0.605 | 0.62 | 0.448 |
| 2 | Original | Half | 15 | Liv | 838 | 838 | 0.99 | 0.999 | 0.996 | 0.814 |
| 2 | Original | Half | 15 | Tum | 838 | 1916 | 0.709 | 0.211 | 0.244 | 0.0823 |
| 3 | Aug | Half | 15 | All | 838 | 3031 | 0.569 | 0.527 | 0.514 | 0.382 |
| 3 | Aug | Half | 15 | Liv | 838 | 883 | 0.876 | 0.967 | 0.974 | 0.748 |
| 3 | Aug | Half | 15 | Tum | 838 | 2148 | 0.263 | 0.0863 | 0.0547 | 0.0163 |
| 4 | Norm | Half | 15 | All | 838 | 2754 | 0.804 | 0.586 | 0.599 | 0.437 |
| 4 | Norm | Half | 15 | Liv | 838 | 838 | 0.959 | 0.995 | 0.994 | 0.815 |
| 4 | Norm | Half | 15 | Tum | 838 | 1916 | 0.649 | 0.177 | 0.204 | 0.0598 |
| 5 | Aug+Norm | Half | 15 | All | 936 | 3175 | 0.633 | 0.563 | 0.554 | 0.427 |
| 5 | Aug+Norm | Half | 15 | Liv | 936 | 936 | 0.853 | 0.964 | 0.964 | 0.81 |
| 5 | Aug+Norm | Half | 15 | Tum | 936 | 2239 | 0.414 | 0.162 | 0.144 | 0.044 |

Table 1: Comparison of Experiments - Training Phase

| Exp | Info | Data | Eps | Cls | Img | Lbl | P | R | mAP@.5 | mAP@.5:.95 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Original | All | 30 | All | 2157 | 4188 | 0.858 | 0.711 | 0.74 | 0.526 |
| 1 | Original | All | 30 | Liv | 2157 | 2157 | 0.953 | 0.945 | 0.965 | 0.806 |
| 1 | Original | All | 30 | Tum | 2157 | 2031 | 0.764 | 0.477 | 0.516 | 0.245 |
| 1.1 | Original | All | 15 | All | 2157 | 6019 | 0.771 | 0.723 | 0.721 | 0.524 |
| 1.1 | Original | All | 15 | Liv | 2157 | 2157 | 0.941 | 0.937 | 0.966 | 0.847 |
| 1.1 | Original | All | 15 | Tum | 2157 | 3862 | 0.6 | 0.509 | 0.475 | 0.202 |
| 2 | Original | Half | 15 | All | 838 | 2754 | 0.836 | 0.605 | 0.619 | 0.448 |
| 2 | Original | Half | 15 | Liv | 838 | 838 | 0.971 | 0.999 | 0.995 | 0.814 |
| 2 | Original | Half | 15 | Tum | 838 | 1916 | 0.701 | 0.21 | 0.243 | 0.0819 |
| 3 | Aug | Half | 15 | All | 838 | 3031 | 0.748 | 0.604 | 0.603 | 0.427 |
| 3 | Aug | Half | 15 | Liv | 838 | 883 | 0.916 | 0.958 | 0.955 | 0.771 |
| 3 | Aug | Half | 15 | Tum | 838 | 2148 | 0.56 | 0.25 | 0.25 | 0.0835 |
| 4 | Norm | Half | 15 | All | 838 | 2754 | 0.787 | 0.622 | 0.627 | 0.453 |
| 4 | Norm | Half | 15 | Liv | 838 | 838 | 0.97 | 0.998 | 0.995 | 0.819 |
| 4 | Norm | Half | 15 | Tum | 838 | 1916 | 0.603 | 0.246 | 0.26 | 0.0861 |
| 5 | Aug+Norm | Half | 15 | All | 936 | 3175 | 0.942 | 0.443 | 0.458 | 0.284 |
| 5 | Aug+Norm | Half | 15 | Liv | 936 | 936 | 0.883 | 0.865 | 0.896 | 0.564 |
| 5 | Aug+Norm | Half | 15 | Tum | 936 | 2239 | 1 | 0.0 | 0.0189 | 0.00493 |

Table 2: Comparison of Experiments - Test Phase

Precision (P) and Recall (R) are evaluation metrics commonly used in object detection tasks.

Precision (P) measures the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives + false positives). In other words, it quantifies the accuracy of the model's positive predictions.

Recall (R) measures the proportion of correctly predicted positive instances (true positives) out of all actual positive instances (true positives + false negatives). It quantifies the ability of the model to capture all positive instances in the dataset.

mAP@.5 evaluates the mean average precision at an IoU threshold of 0.5. It calculates the average precision for each class and then takes the mean across all classes. Intersection over Union (IoU) is a metric used to evaluate the overlap between two bounding boxes. It measures the ratio of the intersection area between the predicted and ground truth bounding boxes to the union area of both boxes. Average precision is a measure of the model's accuracy in predicting both the bounding box location and the class label of an object.

mAP@.5:.95 is similar to mAP@.5, but it considers a range of IoU thresholds from 0.5 to 0.95. It provides a more comprehensive evaluation by considering different levels of overlap between predicted and ground truth bounding boxes.

When each of these metrics (P, R, mAP@.5, mAP@.5:.95) is high, it indicates a high-performance object detection model. A high precision means that the model has a low false positive rate, making accurate positive predictions. A high recall indicates that the model captures a large portion of the true positive instances, minimizing false negatives.

High mAP@.5 and mAP@.5:.95 values imply that the model consistently achieves accurate detections across different IoU thresholds. This demonstrates the model's ability to precisely locate objects and assign correct class labels.

Overall, when precision, recall, mAP@.5, and mAP@.5:.95 are high, it signifies that the model has a good balance of accuracy and completeness in object detection, making it highly effective in identifying and localizing objects in the given dataset.

# 8 Results Discussion

In this section, we discuss the results obtained from the various experiments conducted. Several key insights can be drawn from the results:

- **Effect of Data Augmentation:** Comparing Experiment 2 (without augmentation) and Experiment 3 (with augmentation), we observe that the performance on the 'Liver' and 'Tumor' classes seems to have decreased. The limited amount of data augmentation images possible to create, due to limited resources, may have hindered the model's ability to generalize to different tumor variations. Further experiments with increased and diverse augmentation could help address this limitation and improve accuracy.
- **Effect of Normalization:** Comparing Experiment 2 (without normalization) and Experiment 4 (with normalization), there is a small improvement in the performance metrics for the 'Liver' class. However, the performance on the 'Tumor' class has decreased. This suggests that normalization can help improve the model's performance on certain classes, but it might not always lead to better performance for all classes. The type of normalization applied could be further investigated to optimize the results.
- **Performance Across Classes:** The model's performance varies significantly across different classes. It performs best on the 'Liver' class, with high precision, recall, and mAP scores across all experiments. However, the performance on the 'Tumor' class is significantly lower. This could be due to the inherent difficulty of the task, or it could suggest that the model needs to be further optimized for this specific class.
- **Transforming 3D images to 2D:** To provide the model with input images, we had to slice each 3D image into 2D slices, where each value along the z-axis corresponded to a slice. However, this slicing process could potentially result in data loss (the network now don't have the information about the z axis), particularly for small tumors that are challenging to detect. It is possible that better results could have been achieved by using a different architecture, built in for 3D images directly.

# 9 Results conclusions

In conclusion, these experiments highlight the importance of data preprocessing techniques such as augmentation and normalization in improving model performance. However, they also underscore the need for careful tuning of these techniques, as they can have varying effects on different classes.

When it comes to achieving final results, 15 epochs and training on only half of the data may not always be sufficient. However, our observations revealed that the impact of the number of epochs was less significant compared to the influence of the training data size. Moreover, this approach can still provide insights into the relative performance of different model parameters and techniques. By comparing the results obtained under these conditions, we can make informed decisions about which approaches are more promising and worth further exploration.

It's important to consider the limitations of our computational resources, particularly the low compute power. Training the model for 30 epochs on the entire dataset, which consists of 21,000 slices (including liver, tumor, background) can be a time-consuming process, taking approximately 5 hours to complete. Given these constraints, conducting a comprehensive evaluation on the full dataset might not be feasible within the available time and resources. Therefore, the initial experimentation with 15 epochs and a reduced dataset size serves as a practical compromise to gain preliminary insights and guide our decision making process.

For further experiments, it's beneficial to conduct more extensive studies using augmented data. For instance, augmenting each data point with two additional images and applying normalization to the entire dataset would effectively triple the amount of data and especially could help detecting small tumors. However, implementing such an approach would require additional computational resources and necessitate more workers and stronger computing power to handle the augmented dataset effectively.

# 10 Conclusion

In conclusion, this exercise provided us with valuable insights into the challenges and complexities of detecting liver tumors in medical images. Through data exploration, preprocessing, augmentation, and hyperparameter tuning, we were able to develop an effective model for liver tumor detection. Despite the resource constraints, we believe that our approach has potential for further improvement and could be a valuable tool in the field of medical imaging and diagnosis. The process of refining and optimizing our model underscored the importance of a systematic and iterative approach to machine learning model development.

# 11 Code

DETR - You can access our public Colab notebook here

YOLO - You can access our public Colab notebook here

# References


[1] Patrick Bilic et al. "The Liver Tumor Segmentation Benchmark (LiTS)". In: *Medical Image Analysis* 84 (Feb. 2023), p. 102680. DOI: 10.1016/j.media.2022.102680. URL: https://doi.org/10.1016/j.media.2022.102680.

[2] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/abs/1506.02640.

[3] Nicolas Carion et al. "End-to-End Object Detection with Transformers". In: (2020). DOI: 10.48550/ARXIV.2005.12872. URL: https://arxiv.org/abs/2005.12872.