Liver Segmentation Report

Advanced Medical Image Processing

Tom Mahler[1] and Amir Zaltzman[2]
[1]tom@mahler.tech
[2]amirzaltzman@mail.tau.ac.il

Department of Software and Information Systems Engineering,
Ben Gurion University of the Negev, Israel

Department of Electrical Engineering
Tel Aviv University, Israel

June 5, 2019

1 Introduction

A fully automatic technique for segmenting the liver and localizing its unhealthy tissues is essential in many clinical applications, such as pathological diagnosis, surgical planning, and postoperative assessment. However, it is still a very challenging task due to the complex background, fuzzy boundary, and various appearance of both liver and liver lesions.

1.1 Tasks

In this project we will develop automatic algorithms to segment liver and liver lesions in abdominal CT scans. To achieve this goal we'll utilize the power of deep learning algorithms.

The project consists of the following tasks:

1. Liver segmentation
2. Lesions segmentation

1.2 Data

The training data set consists of 11 CT scans:

- `Data` directory contains CT images converted to `png` format
- `Segmentation` directory contains segmentation masks:
  - Pixels with value 127 indicate liver
  - Pixels with value 255 indicate liver lesion

The data can be downloaded from the following link: data link.

1.3 Evaluation metrics

The following metrics will be used to evaluate the segmentation accuracy:

- Dice similarity coeffcient (Dice):

  $$Dice = \frac{2 \times TP}{2 \times TP + FP + FN}$$

  TP, FP, and FN denote the number of true positive, false positive, and false negative pixels respectively. Dice computes a normalized overlap value between the produced and ground truth segmentation.
- Precision (positive predictive rate):

  $$Precision = \frac{TP}{TP + FP}$$

  Precision expresses the proportion between the true segmented pixels and all pixels the model associates with liver / lesions.
- Recall (sensitivity):

  $$Recall = \frac{TP}{TP + FN}$$

  Recall expresses the proportion between the true segmented pixels and all liver / lesions pixels.

2 Installation

2.1 `TensorFlow 2.0`

In this work, we are making use of `TensorFlow 2.0` API on `Google Colab` notebook set on GPU runtime (GPU: 1xTesla K80, having 2496 CUDA cores, compute 3.7, 12GB (11.439GB Usable) GDDR5 VRAM).

2.2 Download the Dataset

The dataset was uploaded to `MahlerTom/LiverSegmentation`, so we first need to clone the repository, with the data. To make things easier, we also define:

- `repo_path` – the repository path (this should be cross platform since we use `os` module)
- `train_path` – the train dataset path
- `val_path` – the test dataset path

3 Preparing the Dataset

Before we begin our training, we need to prepare the dataset. Since we are using `TensorFlow 2.0`, we will make use of its functions. We followed the guide at:
`https://www.tensorflow.org/alpha/tutorials/load_data/images`
Tensorflow makes use of smart functions that can load images given their paths. The data structure is as follows:

```
(img_path, mask_path)
```

3.1 Load and Preprocess images using `TF 2.0`

First, we will load the data sets to our training and testing sessions, using our implemented function `load_data`. Then, we apply the following preprocess acts:

- Normalize grayscale values from 0-255 to 0-1.
- Resize input images shape from 512×512 to 256×256.
- Divide the data into 3 datasets: train, validation and test.

3.2 Basic dataset manipulations before training

To train the model with this dataset we will decide:

- If and how to shuffle the data.
- How to divide to batches.

Furthermore, we will want the training process:

- To be repeatable over the the dataset.
- To make the batches available for training as soon as possible.

These features can be easily added using the `tf.data` API.

4 Define the model

4.1 Model

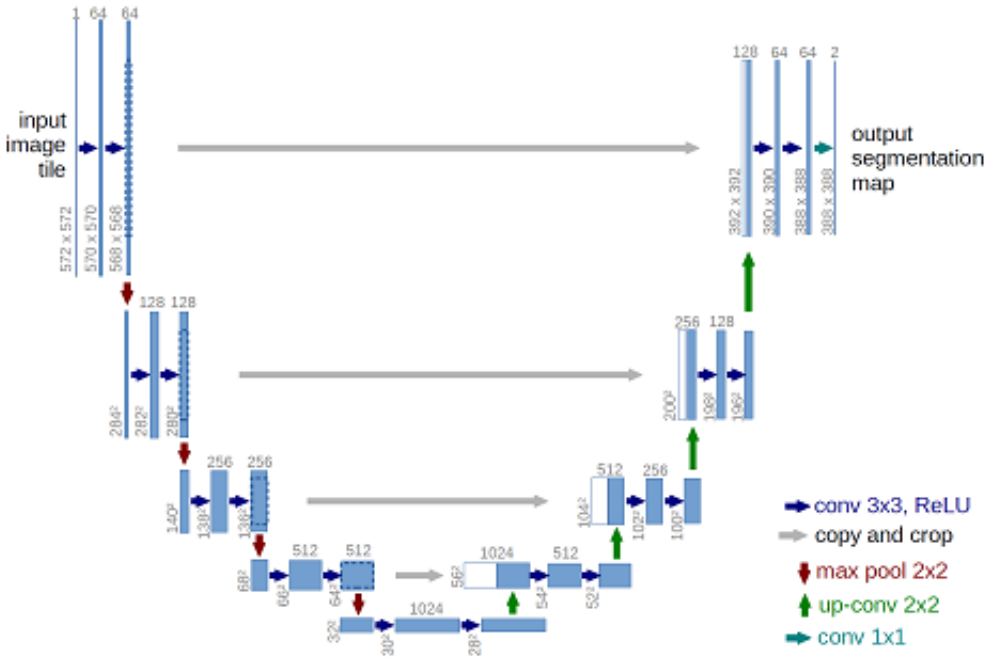
To create segmentation masks of liver and lesions for CT scans, we will use the U-net training with few variations.

U-net, in short, is a state-of-the-art network aimed for segmentation tasks.

The U-net architecture contains two paths:

1. First path is the contraction path (also called as the encoder). It is used to capture the context in the image. The encoder is constructed from CNN and max-pooling layers.
2. The second path is the symmetric expanding path (also called as the decoder). It is used to enable precise localization using transposed convolutions.

The model architecture is shown below:



In our implementation, we enabled configurations of the training model input parameters – We chose the combination of the following parameters to achieve best results (with our GPU):

Parameters	Value
Learning rate	10^{-3}
Batch size	32
Image input resolution	256×256px
# of initial filters	4
# of epochs	30
Optimizer	Adam

4.2 Define matrices and loss function

We define the following **matrices scores** regarding our model, to evaluate the quality of the results:

Dice coef total	dice coefficient score for both liver and lesions segmentation
Dice coef liver	dice coefficient score for liver segmentation only
Precision liver	Precision score for liver segmentation only
Recall liver	Recall score for liver segmentation only
Dice coef lesions	dice coefficient score for lesions segmentation only
Precision lesions	Precision score for lesions segmentation only
Recall lesions	Recall score for lesions segmentation only

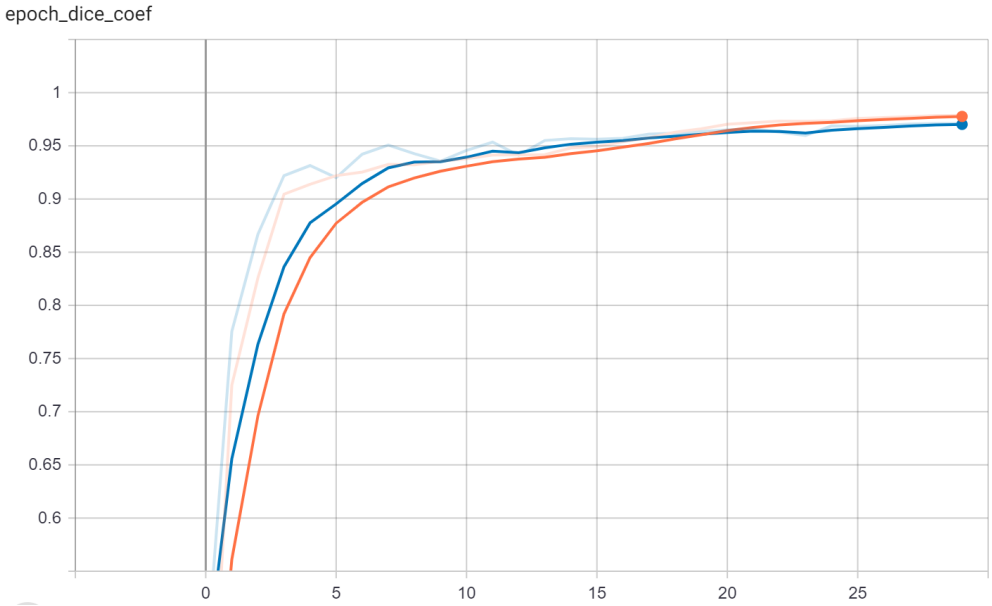
The model will be defined with the **loss** of $1 - \mathit{Dice\ coef\ total}$. We use this loss definition because it enables the training process find hidden features of liver and lesions simultaneously. This loss decrements improves the segmentation task each epoch.

Note: matrices scores are calculated with quantified predicted images.

5 Train Model

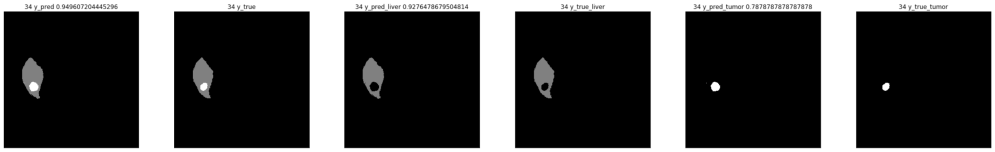
To train the model above, we implemented the following functions:

- `model_fit` – receives as input the parameters (e.g., number of epochs, learning rate, batch size, etc.). It performs the training session over the train and val datasets (as described in "Define the model" section 4).



Example of *Dice coef total* vs. # of epochs, for train (orange) and val (blue) datasets

- `predict_model` – Plots row of 4 figures for each CT scan in the dataset (in this case: val dataset):
  1. Continuous predicted mask (continuous grayscale image).
  2. Quantified predicted mask (3-level grayscale image).
  3. Original ground-truth mask (3-level grayscale image).
  4. Original CT scan image (continious grayscale image).
- `print_model_score` – Plots row of 6 figures for each CT scan in the dataset:
  1. Quantified predicted mask and its *Dice coef total* score.
  2. Original ground-truth mask.
  3. Quantified predicted **liver** mask only and its *Dice coef total* score.
  4. Original ground-truth **liver** mask only.
  5. Original ground-truth **lesion** mask only and its *Dice coef total* score.
  6. Original ground-truth **lesion** mask only.



Example

- `print_model_score_table` – Prints table of the following categories:

```
| ID | Image  | Dice All | Dice Liver | Percision Liver | Recall Liver | Dice Tumor | Percision Tumor | Recall Tumor |
```

6 Experiments

In this section, we test our methods with different parameters to find the best ones. There are numerous parameters to choose from, and we will not try them all. We shall focus on the following parameters, which are the most relevant ones with respect to the implications on the training and results:

Parameters
Learning rate
Batch size
Image input resolution
of initial filters

In addition, we define several experiments using different parameters:

Learning Rate	Batch Size	Image Resolution	# of Filters	Shuffled
$\{5 * 10^{-4}, 10^{-3}, 5 * 10^{-3}\}$	32	256×256	4	$\{True, False\}$

6.1 Shuffled vs Not Shuffled

In this section we show the results of the training the model on a shuffled vs on a not shuffled dataset. Usually, shuffling the data is recommended.

6.2 Remove Images

As we can see, the first two patients are not so good, so we shall try to remove them:

0	1	2	3	4	5	6	7	8	9	10
28	27	130	164	240	162	174	167	167	162	116

We can notice that the first 2 patients (numbered 0 and 1) have the least amount of CT scans. Moreover, after reviewing manually their scans, their quality image is poor. We decided to check if we omit these scans from our training, we could have perform better segmentation results.

6.3 Data Augmentation

We tried to augment new scans out of our existing training dataset. Our methods to augment new scans were:

- Rotate images by 90°, 180° and 270°
- Adjust image brightness ± 0.05.

Run	Name	Optimizer	Batch Size	Resize	Filters	Learning Rate
0605-180358		Adam	32	256×256	4	0.001
0605-180456	shuffled	Adam	32	256×256	4	0.001
0605-180517	not_shuffled	Adam	32	256×256	4	0.001
0605-180538	lr5e-3	Adam	32	256×256	4	0.005
0605-180555	lr5e-4	Adam	32	256×256	4	0.0005
0605-180618	remove_shuffle	Adam	32	256×256	4	0.001
0605-180636	da	Adam	32	256×256	4	0.001
0605-180653	da_5e-4	Adam	32	256×256	4	0.0005
0605-180713	da_remove	Adam	32	256×256	4	0.001

Loss	Dice Coefficient	Precision	Recall	Dice Coefficient Liver	Dice Coefficient Tumor
3.663%	96.337%	99.530%	57.927%	94.502%	5.386%
4.395%	95.605%	98.822%	85.7%	94.099%	2.950%
15.447%	84.553%	90.328%	85.465%	83.459%	0.201%
99.799%	0.201%	0%	0%	0.209%	0.201%
14.633%	85.367%	95.099%	76.411%	86.026%	0.201%
4.253%	95.747%	98.654%	78.65%	94.212%	3.678%
7.838%	92.162%	97.598%	89.396%	94.205%	0.201%
11.345%	88.655%	97.158%	89.322%	93.158%	0.201%
35.396%	64.604%	94.673%	7.920%	59.793%	0.201%

7 Conclusions and Future Work

In this project, we have implemented a U-Net network architecture for liver and tumor segmentation. We have tested nine different configurations of our base model. In most of our attempts, segmenting the liver gave very good results. However, segmenting the lesions was a much more challenging task. Only two configurations (0605-180358 and 0601-180618) could segment the lesions well in most CT scans.

As future work to improve the results of this project, we would take the following steps:

1. Use a different loss function, perhaps a loss function of the lesion dice coefficient, since the models with the highest tumor dice coefficient gave the best results for segmenting the lesions.
2. Using different data augmentation configurations, such as small rotations instead of 90°, 180°, and 270°.
3. Determine a threshold for minimal size lesions to adjust our task for realistic cases, consulting with experts.
4. Train the model with a variable batch size, and in each batch to include all images of a single patients.
5. Attempt to use a different network, perhaps an LSTM, which can learn relations from a sequence of scans instead of from each scan individually. By doing so, we may increase the chances of detecting a tumor in subsequent scans.