POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

MRI Organ Segmentation

Biomedical Computer Vision Course
Project Report

Author: **Stefano Arcaro**

Student ID: 10685760
Academic Year: 2023-24

Abstract

This project, part of the Biomedical Computer Vision course, aims to address the task of organ segmentation, given apatient's MRI scan using deep learning techniques.

The dataset used is from the CHAOS 2019 challenge, which includes MRI volumes acquired with T1-DUAL (InPhase) and T2-SPIR sequences for 40 patients, together with the corresponding ground truth segmentations of four vital organs - liver, spleen, and the two kidneys - for each slice of the DICOM volumes.

The project focuses on the training and testing of a famous deep learning model - the UNet - implemented with TensorFlow and Keras.

This report begins with the exploration of the CHAOS 2019 dataset, followed by the description of the approach used to solve the problem, from data preprocessing, to the chosen model architecture and performance metrics. Additionally, data augmentation techniques using the Albumentations library are employed as as part of the experiments.

Finally, the report presents and analyzes the results of the previously described experiments, and addresses the challenges encountered during the project.

An extra section describes the small Web Application created to visualize the work of the best model on a user-uploaded DICOM volume.

Contents

1 | Dataset Exploration

The dataset used for this project is the one from the CHAOS challenge, held in The IEEE International Symposium on Biomedical Imaging (ISBI) in 2019.

The challenge consisted of several tasks: the focus of this project was the one concerning the segmentation of liver, spleen, and the two kidneys, using only the Magnetic Resonance Imaging (MRI) datasets, acquired with two different sequences - namely, T1-DUAL and T2-SPIR - and saved in the DICOM format.

For each of the two sequences, the dataset is composed of the volumes of 40 patients, separated into 20 for training, and 20 for testing. As I have no medical expertise, and the patients volumes used for testing are not provided together with their ground truth segmentations, the part of the dataset I was able to use for this project consists of the 20 patients provided for training.

To sum up, each patient has a MRI volume, and the corresponding ground truth segmentations for each slice of the volume, for each of the two sequences (T1-DUAL and T2-SPIR). Every patient's volumes have different slice and voxel dimensions, as well as different numbers of slices. This makes it quite a complex dataset to use, and significant pre-processing of the data is required.

2 | Data Pre-processing

As mentioned before, each patient's volume is different from the others. This poses a problem, as the input of the model will need to have fixed dimensions: therefore, I first computed the average dimension for the slices, together with the average number of slices, across all patients. Then, each volume was preprocessed in the following way:

- The voxel dimensions for each patient were normalized, meaning they were brought to a target resolution of [1, 1, 1]
- Each volume was reshaped to have the average number of slices
- Each slice was reshaped to have the average width and height

This preprocessing was tricky, mainly due to the need of interpolation to modify the number of slices of each volume without ruining the dataset.

The last step simply consisted of vertically stacking all the samples, which were at this point images with fixed width and height.

3 | Experiments

For each experiment, the data was split into training, validation, and test sets. This was done using the same seed to later be able to better compare the experiments results.

Each model was trained on the training set, and used the validation set to fine-tune the models hyperparameters. Due to time constraints, it was unfortunately not possible to perform cross-validation for the experiments, which would have given a better estimate of the models performances.

The performance of each of them was then measured on the test set.

The performance metric used to validate and test the model is the Mean Intersection over Union (MeanIoU), a custom metric that works quite well for segmentation tasks.

3.1. Hyperparameters

Given the experiments were all done using the same model - UNet - and they only varied in the data augmentation part of the process, the best hyperparameters I found were the same for each experiment.

They are as follows:

- Learning rate = 1e-3
- Batch size = 8
- Optimizer = Adam

Two callbacks were also used, both monitoring the validation MeanIoU:

- Early Stopping, with patience = 30
- Reduce LR On Plateau, with patience = 25

3.2. UNet Model - No Data Augmentation

The first, and simplest, approach was to create a UNet model and fit it to the preprocessed data without any augmentation pipeline. This was also done to obtain a baseline, in order to evaluate the impact of data augmentation for this task.

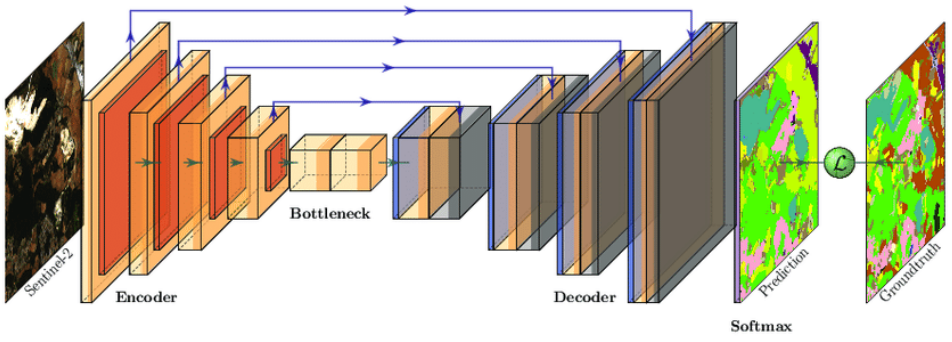


Figure 3.1: Architecture of the UNet model.

3.3. UNet - Heavy Data Augmentation

To perform data augmentation in a segmentation task, one must be careful to correctly augment the ground truth mask as well. This is why I chose to use the Albumentations library - imported as A - which makes this process very easy.

The first attempt at training the model with augmented data was with quite the aggressive augmentation pipeline, as can be seen below:

```python
# Define a function to convert images to uint8 and back to float32
def to_uint8(image, **kwargs):
    return (image * 255).astype('uint8')

def to_float32(image, **kwargs):
    return image.astype('float32') / 255.0

# Create the data augmentation pipeline
augmentation_pipeline = A.Compose([
    A.Rotate(limit=15, p=0.5),
    A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
    A.Lambda(image=to_uint8),
    A.Equalize(p=0.5),
    A.Lambda(image=to_float32),
    A.GaussNoise(var_limit=(0.0, 10.0), p=0.5),
    A.Blur(blur_limit=(3,5), p=0.5)
])
```

Listing 3.1: Heavy data augmentation pipeline.

3.4. UNet - Light Data Augmentation

To check how much the heaviest parts of the first data augmentation pipeline - namely, Gaussian Noise and Blur - were affecting the training of the model, I also tried removing them, while keeping the rest of the previous pipeline.

The code is as follows:

```python
# Define a function to convert images to uint8 and back to float32
def to_uint8(image, **kwargs):
    return (image * 255).astype('uint8')

def to_float32(image, **kwargs):
    return image.astype('float32') / 255.0

# Create the data augmentation pipeline
augmentation_pipeline = A.Compose([
    A.Rotate(limit=15, p=0.5),
    A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
    A.Lambda(image=to_uint8),
    A.Equalize(p=0.5),
    A.Lambda(image=to_float32)
])
```

Listing 3.2: Light data augmentation pipeline.

4 | Results

Quite surprisingly, the results were not quite what I had expected.

While I expected the models trained on augmented data to outperform the baseline by a significant margin - significant meaning at least a couple percentage points above - the results were peculiar.

As can be observed in Table 4.1, the best performing model was actually the baseline. In this case, augmenting the data actually hurt the models performances.

Project Results

	UNet Base	**UNet - Heavy DA**	**UNet - Light DA**
Train MeanIoU	0.8426	0.6158	0.8375
Test MeanIoU	0.8075	0.7919	0.8030

Table 4.1: Comparison of the trained models results.

One thing to note, is the difference between the model with an aggressive data augmentation pipeline, and the other two models: while the other two saw some slight overfitting (see Figure 4.1), said model performed quite badly on the augmented training set, while still being able to generalize and almost reach the performance of the other models.

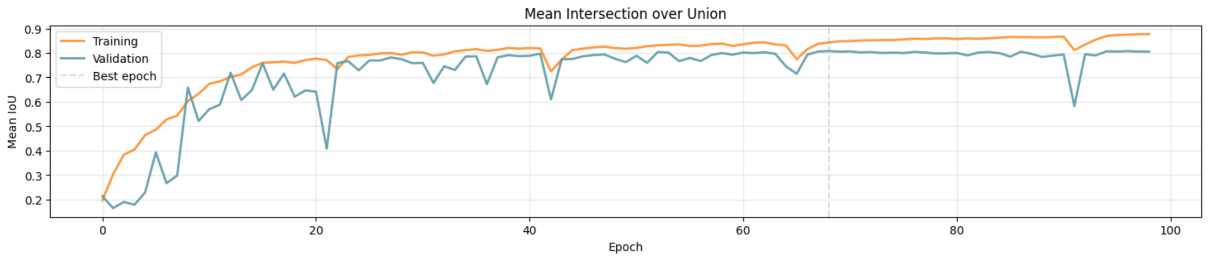
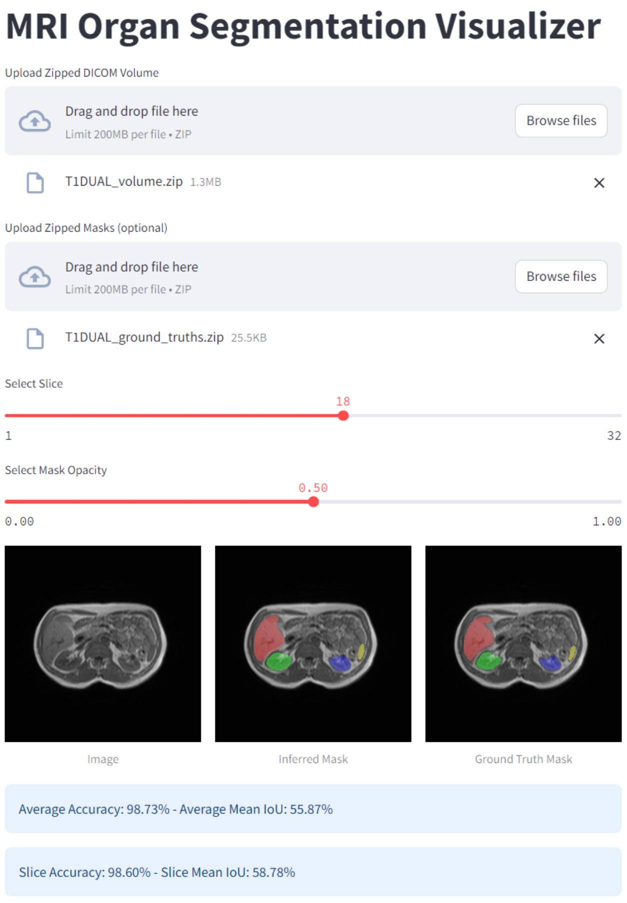


Figure 4.1: Training and validation performance of the best model.

5 | Visualization Web App

As a little extra for this project, I developed and deployed a small web application using Streamlit. The purpose of this application is to display and compare the model's inferred segmentations with the ground truth segmentations, or, in their absence, just check the model's performance.

The user is able to upload a zipped folder with a patient's MRI volume, in DICOM format, as well as the ground truth segmentations for the selected volume, if available. After the model is done with the inference, the user can visualize the segmentations with an interactive slider to select the desired slice.


MRI Organ Segmentation Visualizer

Upload Zipped DICOM Volume

Drag and drop file here
Limit 200MB per file • ZIP
Browse files

T1DUAL_volume.zip 1.3MB

Upload Zipped Masks (optional)

Drag and drop file here
Limit 200MB per file • ZIP
Browse files

T1DUAL_ground_truths.zip 25.5KB

Select Slice: 18 (1–32)

Select Mask Opacity: 0.50 (0.00–1.00)

Image | Inferred Mask | Ground Truth Mask

Average Accuracy: 98.73% - Average Mean IoU: 55.87%

Slice Accuracy: 98.60% - Slice Mean IoU: 58.78%

Figure 5.1: The visualization web app.

6 | Conclusion

In conclusion, this Biomedical Computer Vision project successfully addressed the challenge of organ segmentation in MRI images through the implementation of the UNet architecture. The model, trained on the CHAOS 2019 dataset, demonstrated promising results in accurately segmenting vital organs such as the liver, spleen, and kidneys.

Throughout the project, significant challenges were met, including managing complex biomedical data volumes and selection of data augmentation techniques - the latter can definitely be improved upon.

Further work could be done under several aspects, such as extending the model to new datasets, and exploring transfer learning and fine-tuning approaches.

In summary, this project concludes my journey in the Biomedical Computer Vision course, which has been an absolutely enriching experience.