Coursework II 2020/2021

Released: Friday November 13th 2020
Deadline: Friday December 14th 2020 at 17:00
Weight: 25 %

The module is assessed by 50% examination and 50% continuous assessments (aka two courseworks). This document specifies the second coursework. The objective of this coursework is for you to gain practical experience in designing, implementing, training and optimizing a neural network to carry out a specific real-world task. The coursework is designed as a competition on Kaggle where you will work together in groups, each group submitting a solution.

You should also submit a report describing how you have done the coursework via Canvas within the deadline by the group leader. If you miss this deadline, your mark will be reduced by 5% (out of 100%) for each School working day (or part thereof) your submission is late. Feedback with marks will be returned within three weeks (excluding Christmas holiday) of the hand-in deadline.

1. Introduction

The task of this year is about semantic segmentation of magnetic resonance (MR) images using deep learning. In digital image processing and computer vision, semantic image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. This process can simplify and/or change the representation of an image into something that is more meaningful and easier to analyze.

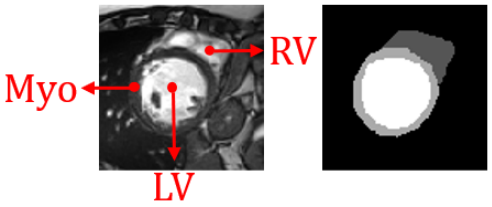

Figure 1: A cardiovascular MR image (left) is segmented into a mask image on the right. The segmented mask includes four regions: the background region (black), the left ventricle (LV) region (white), the myocardium (Myo) region (white gray) and the right ventricle (RV) region (dark gray).

Specifically, we are concerned with semantic image segmentation of cardiovascular MR (CMR) images. CMR imaging is the gold standard for assessing cardiac chamber volume and mass for many cardiovascular diseases. For decades, clinicians have been relying on manual segmentation approaches to derive quantitative measures such as left ventricle volume, mass and ejection fraction. However, manual expert segmentation is tedious, time-consuming and prone to subjective errors. It becomes impractical when dealing with large-scale datasets. For this, the aim of this coursework

to develop an automated methodology using Convolutional Neural Network (CNN) that is capable of segmenting a CMR image into four regions: the background, the left ventricle (LV), the right ventricle (RV) and the myocardium (Myo). Figure 1 gives you an example of such a segmentation.

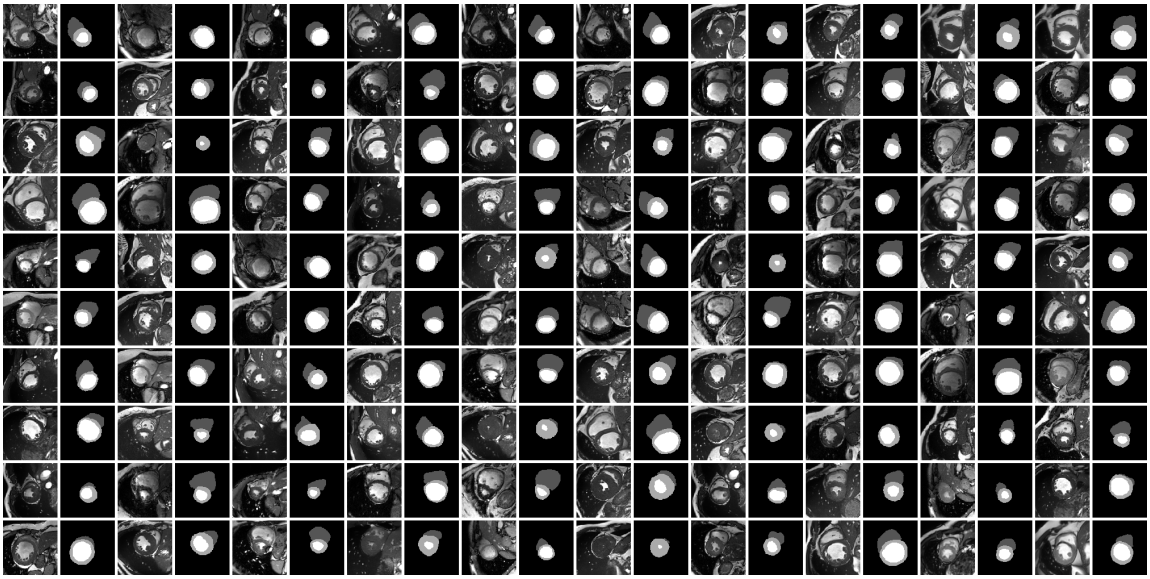

Figure 2: CMR images and their masks in the training set.

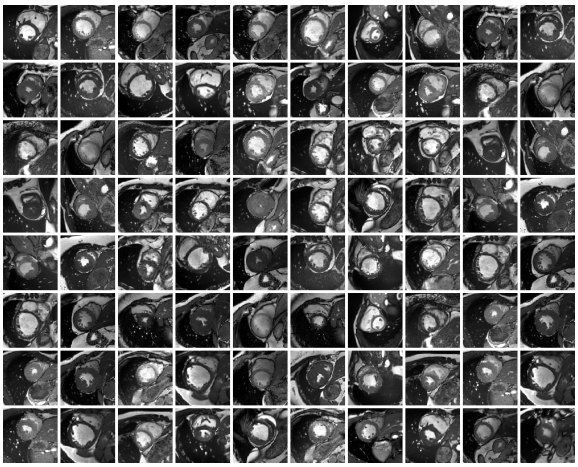

Figure 3: CMR images in the test set.

Dataset: First, we introduce the dataset used for this task. The dataset can be downloaded from the ACDC[1] challenge but we have modified it for the sake of simplicity. The dataset after our modifications contains a totoal of 200 CMR images in a PNG format. We split the data into 50% for training, 10% for validation and 40% for testing. For the training set, there are 100 CMR

[1] https://www.creatis.insa-lyon.fr/Challenge/acdc/

images and 100 respective ground truth mask images. Figure 2 shows all images and their masks in the training set. You should use the images and masks in this set to train your network. For the validation set, there are 20 CMR images and 20 mask images, which look similar to those in the training set. You should use this set to tune the hyperparameters (ie. CNN architectures, loss functions, optimization algorithms, etc) of your method so as to maximize the overall performance. For the test set, there are 80 CMR images but no ground truth masks are given. Figure 3 displays all 80 test images. For the test set, we hold the ground truth masks. ***Your mission is to see how accurately your method can segment these CMR images in the test set, by comparing your segmentation masks with the ground truth masks we hold***.

Network architectures: There are many state-of-the-art CNN architectures for semantic image segmentation. Popular choices are FCN[2], DeepLab[3], SegNet[4], etc. However, these networks may not be directly applicable to this task due to the limitation of computational power you have. It is important you take this into account and design a suitable segmentation network that is capable to achieve a result as accurate as possible. Note: ***Regardless of which network you use, in this task the input tensor to the network must be of size (B, 1, H, W) and the output tensor from the network must be of size (B, 4, H, W). Here B denotes batch size, H and W stand for image height and width, respectively***.

Metric: *Dice similarity coefficient*[5] or simply *Dice* will be used to measure segmentation performance with respect to ground truth. Dice between two binary masks X and Y is defined as the intersection ratio between both the overlap area and the average area of the two masks, which is given as

$$Dice(X, Y) = \frac{2|X \cap Y|}{|X| + |Y|}.$$

Dice varies from 0-1, with high values corresponding to a better match. Note: ***Once you have predicted segmentation masks of images in the test set, you should call the submission script we have provided. It will automatically create a CSV file which encodes your results in the format required by Kaggle. You then submit the CSV file to Kaggle and an average Dice over all three regions for 80 test cases will be automatically computed and ranked there. Check our video for more instructions on this step***.

Loss function: To obtain precise segmentation results, it is important you implement an effective loss function. Typical choices are the cross-entropy loss[6], the soft Dice loss[7], the Focal loss[8], the boundary loss[9], or a combination of some of them.

[2] https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf
[3] https://arxiv.org/pdf/1606.00915.pdf
[4] https://arxiv.org/pdf/1511.00561.pdf
[5] https://en.wikipedia.org/wiki/Sørensen–Dice-coefficient
[6] https://en.wikipedia.org/wiki/Cross_entropy
[7] https://arxiv.org/pdf/1606.04797.pdf
[8] https://arxiv.org/pdf/1708.02002.pdf
[9] https://arxiv.org/pdf/1812.07032.pdf

2. Getting Started

1. Go to Canvas Assignment and download `CW2.zip`. Unzip the files to your home directory (for example it is `/Users/jduan/Desktop/CW2` on my laptop). You should then see `data` and `tutorial.ipynb` in the unzipped folder.

2. After these files are downloaded and unzipped it is fine you develop your method using Jupyter Notebook on your own laptop, but you will need to install `torch` and `opencv` python libraries. Alternatively, we recommend you use Docker where we have installed all dependencies for you. Open the terminal and type the following two command lines

   ▷ `docker pull nc2020/cw2`
   ▷ `docker run -it --rm -v /Users/jduan/Desktop/CW2:/src -p 8888:8888 nc2020/cw2`

   The only thing you need to change above is to replace `/Users/jduan/Desktop/CW2` with the directory on your laptop (for Mac and Linux users only). If you are a Windows user, you should replace the second command above with

   ▷ `docker run -it --rm -v C:\Users\jduan\Desktop\CW2:/src -p 8888:8888 nc2020/cw2`

   Afterwards, copy the link displayed in the terminal onto your browser to start Jupyter Notebook from Docker, then click on `tutorial.ipynb` and complete the tutorial.

   Important notes: (1) the Docker use here is slightly different to lab tutorials. Please follow the instructions here only; (2) because I use `-v` parameter to pass `/Users/jduan/Desktop/CW2` on my local laptop to `/src` in Docker, all files under `/Users/jduan/Desktop/CW2` will be uploaded to `/src` in Docker. It is important you remember to use Docker paths (ie. `/src`) all the time in your code development; (3) these two paths are synchronized. That is to say if you change anything in `/Users/jduan/Desktop/CW2` the content in `/src` will be automatically updated and vice versa. In other words, you do not need to worry about losing your work if you terminate Docker Jupyter Notebook and you have all your work saved locally on your laptop.

3. After your network is trained deploy it on the images in the test set in `/src/data/test/images` and save the predicted segmentation masks to `/src/data/test/masks`. If successful the masks will be also produced in `/Users/jduan/Desktop/CW2/data/test/masks` due to synchronization. You then need to run the submission conversion code in the last part of `tutorial.ipynb` to encode your masks into a CSV file and then submit it to Kaggle for ranking. We have provided a video to explain how the process is done exactly.

4. Write one group report explaining what you did and what you found. Within the report you should insert source code. A reasonable length for the report would be between 2000 and 3000 words excluding the reference list, plus as many diagrams, tables and graphs as you think appropriate. The 25 marks are divided into four sections: training (30%), validation (10%), inference (10%), code quality (10%) and report quality (40%). The first four sections are marked based on your inserted source code and the last section on your written report.

   ▷ For training, you are expected to show the following components: network architecture, loss function, optimizer and training processing. For the training process, you should

show your understanding on the number of epochs required to train your network, as well as data loading. It also necessary to show correct training mode (`model.train`), zero gradient (`optimizer.zero_grad`), backpropagation (`loss.backward`), optimization (`optimizer.step`), etc.

- ▷ For validation, you are expected to show how you use this process to select reasonable hyperparameters in your network.
- ▷ For inference (deployment), you are expected to show how well your trained model has performed on test set on Kaggle. The performance will account for 50% and the remaining 50% comes from your inference code. Ideally, you should save the trained model and then load it for inference.
- ▷ For coding, high quality code should be well structured and commented.
- ▷ For the report, marks are further divided into
  - ▷ Introduction (10%): discuss the data sets involved, the machine learning task, relevant work and what you aimed to achieve.
  - ▷ Implementation (35%): describe how you implemented your neural network and the associated performance analysis mechanisms. Explain why you chose to do it that way. Remember to cite any sources you used.
  - ▷ Experiment (40%): describe the experiments you carried out to optimize your network's generalization performance and present the results you obtained. Explain in detail how you used the training, validation and test data sets. The results should be presented in a statistically rigorous manner.
  - ▷ Conclusion (10%): summarize your key findings, including which factors proved most crucial, and what was the best generalization performance you achieved.
  - ▷ Contribution (5%): description of contribution of each member in the group using peer evaluation CSV file as in CW1.

3. Files to Submit

The group leader should submit on Canvas a HTML file titled group`#`.html, where `#` should be replaced by the group number. More specifically, the HTML file is generated from your Jupyter Notebook (File → Download as → HTML). In your Notebook, you should include source code and write the report with the corresponding section headings as described above.

4. Assessment

The groups will be marked by your HTML report containing source code as indicated by the percentages above. Note that the outcome of the competition will not impact the overall assessment significantly. If there is evidence of significant differences in contributions among group members, then individual marks may be adjusted.

5. Help and Advice

If you get stuck, feel free to ask for advice in Teams or you can talk to us over our office hours or synchronous online sessions. It is highly recommended that you make use of online collaborative tools such as Zoom for this group coursework.