Technical University of Denmark

Image segmentation

Deep learning in Computer Vision

Maiken Errebo Lindberg, s114012
Jules Belveze, s182291

June 21, 2019

Contents

1 Image segmentation

1.1 Introduction

The aim of this project is to perform segmentation of different nuclei in images using deep learning. The dataset of nuclei images are obtained from the Kaggle 2018 Data Science Bowl challenge [1].

1.2 Methods

In image processing, semantic segmentation refers to predicting the class label of the object each pixel belongs to. The process of semantic segmentation of the nuclei images are performed using the architecture of the U-Net to segment the nuclei from the background, while the watershed algorithm is applied to the classification of the U-Net to segment any connected nuclei from each other.

1.2.1 Semantic segmentation using U-Net

The architecture of the U-Net consists of an encoder, a bottleneck and a decoder. The encoder is comprised of a series of steps of downsampling by using max-pooling after each convolutional layer. The image size is reduced by 50 % for each series of convolutional layer and max-pooling. Four such steps are applying leading to reduction of the input image size of 256 by 256 to a size of 16 by 16. The bottleneck convolutional layer basically consists of single convolutional layer, which takes this reduced image as input and its outputs is feed directly into the decoder. On the contrary to the encoder, the decoder consists of a series of upsampling steps, where each unsampling step increases the image size by two in both height and width are followed by a convolutional layer. Such four upsamling steps thus increases the image size from 16 by 16 to the original size of 256 by 256.

1.2.2 Watershed

In generally, the watershed algorithm within image processing for the purpose of image segmentation. The watershed algorithm are applied on grayscale images to separate connecting segments in this case of nuclei. Different principles of the watershed algorithm exist, but in case the principle of topographic distances are employed. This watershed principles works by considering the image pixel intensities as heights and then markers at the local maxima are set, and the distances to the background are computed and watershed lines are drawn at the minimum.

1.2.3 Performance evaluation

The performance of the prediction of the segmented nuclei in each image are evaluated using the mean average precision (MAP) defined as in Eq. 1

$$\text{MAP} = \frac{1}{|\text{threshold}|} \sum_i \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i + \text{FP}_i} \tag{1}$$

where only nuclei with a intersection over union (IoU) of greater than the threshold of 0.8 to the ground truth are considered as correctly segmented nuclei corresponding to true positives (TP), the false negatives (FN) corresponds to true nuclei that has been classified as background and false positives (FP) corresponds to incorrectly predicted nuclei that is actually true background in Eq. 1.

1.2.4 Data preprocessing

The dataset comes from the Kaggle 2018 Data Science Bowl [1] and gathers biological images containing nuclei. For each images a segmented masks of each nucleus is provided. The images are loaded as RGBA images and then converted to RGB, whereas the masks are loaded as RGB and then converted to black and white. For the sake of convenience, for each picture we merge all the masks into a single while attributing a key to each nuclei. In addition, we create an instance of the *Dataset* class of PyTorch in order to load our data and be able to apply it the desired transformations.

1.3 Results

First of all we train our model on RGB images using a dice loss function. It turns out that the loss was really low but the network was predicting all the pixels as background. This is due to the imbalance of our dataset. Indeed, nuclei occupy a relatively small proportion of the image. Thus, by predicting everything as background the network is still performing "well". For this reason, we decided to implement a weighted version of the dice loss, that will penalize more the background class. Figure 1 depicts the learning curve using both unweighted and weighted loss function for gray-scale images.

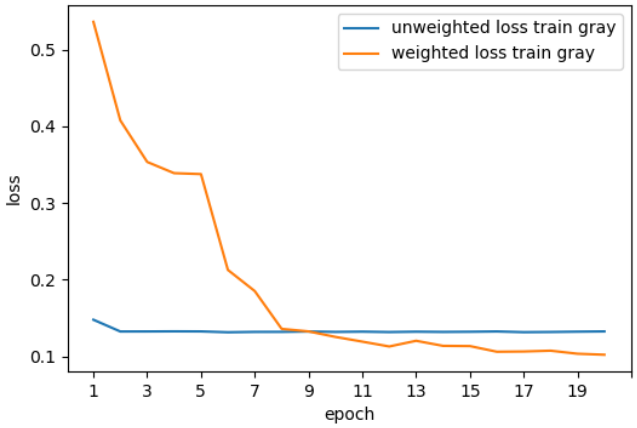

Figure 1: Evolution of the loss function with respect to the number of epochs for a weighted and unweighted dice loss using grayscale images as input to U-Net.

As mentioned above, this figure pinpoints the fact the network always predicts pixels as background when using the unweighted loss function. For this reason the associated learning curve remains steady. In opposition, we can see that the use of a weighted loss function leads to an expected learning curve, and finally reaches of a training loss of 10.12. For this reason we will keep this function for the rest of the analysis.

Figure 2 shows the different learning curve using RGB images and gray-scale images. As we can see when working with RGB images, the model stops learning really early, after 4 epochs. Whereas it keeps learning when using gray-scale images.

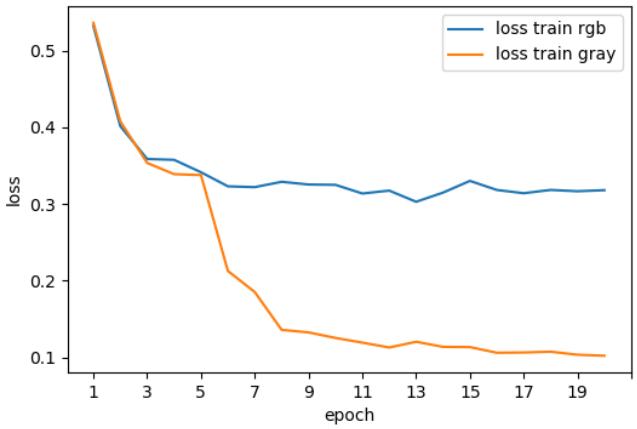

Figure 2: Evolution of the loss function with respect to the number of epochs using a weighted dice loss, and using either grayscale or RGB images as input to the U-Net.

The performance of the U-Net has also been evaluated using different approaches of data augmentation. In Figure 3, the loss function for different transforms of the input images are shown. Hereof it is clear that the loss function from the non-transformed input image is much lower than when transformation of the input images are applied. Therefore we will use this input for the following analysis.

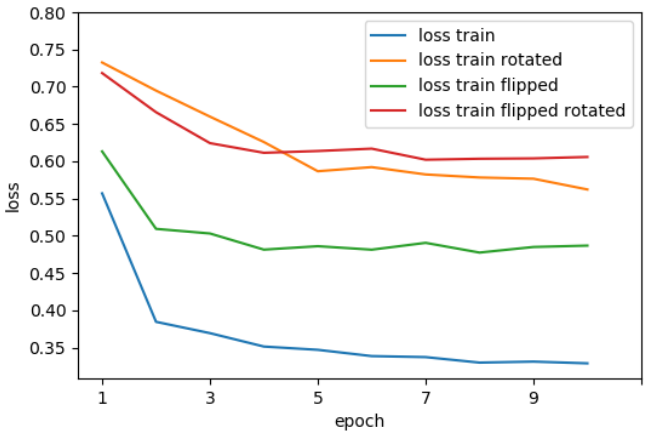

Figure 3: Evolution of the loss function with respect to the number of epochs for different transformations of the input images using a weighted dice loss.

We now evaluate the different models set up using the mean average precision metric, described in 1.2.3, for different IoU thresholds. The results are reported in Table 1.

Threshold	0.4	0.5	0.6	0.7	0.8
MAP gray normalized	0.54	0.46	0.40	0.35	0.26
MAP gray	0.40	0.34	0.30	0.20	0.07
MAP RGB normalized	0.48	0.42	0.36	0.28	0.13
MAP RGB	0.46	0.41	0.35	0.28	0.14

Table 1: Mean average precision (MAP) for different IoU thresholds.

Intuitively the mean average precision is decreasing with the threshold. The best performance is reached when using gray scale normalized images. Since our two models using gray scale images are trained on the same number of epochs it is not surprising that normalizing the input leads to a better performance. Indeed, normalizing helps to get the within a range, which helps in making training faster. However, it is surprising that normalizing the RGB images does not affect much the performance.

For a threshold of 0.8 our best performance is a mean average precision of 0.26. This low performance can be explain by the poor results we got using the watershed algorithm for instance segmentation. As depicted in Figure 4 when two nuclei are really close to each other the network often detects a bigger single nucleus which is the result of the combination of the two. This is the reason why we used the watershed algorithm to then segment them. It turned out that the algorithm is not performing as expected.

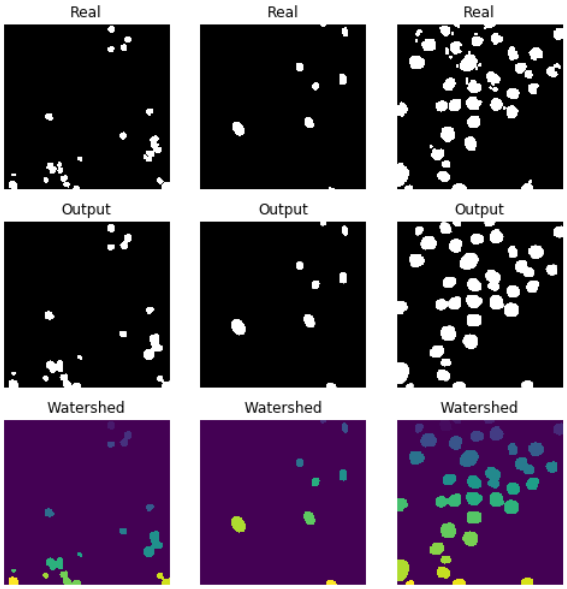

Figure 4: Expected mask, predicted mask and segmented mask after using watershed for three different images.

1.4 Discussion

1.4.1 Open topic: Class imbalance

Within machine learning, class imbalance is a very big issue. This issue with class imbalance is also very evident in results obtained from U-Net, when using an unweighted loss function. As described in Section 1.3 and inspected from Figure 1 the use of the unweighted loss function leads to that all pixels are predicted as black, i.e. background, since the majority class is background and hence it will thus give a very good accuracy of the network, but will lead to a high number of false negatives (FN), when such a prediction is made. In real life, the presence of FN usually has much higher cost than the presence of false positives (FP). An example from real life is for instance when making predictions about whether a patient has cancer or not. In this case, FN would mean that the patient is incorrectly predicted as being healthy, when the patient actually has cancer, which commonly has fatal consequences. When the contrary is true, the patient would incorrectly be diagnosed with cancer even though the patient is healthy, which may lead to that the patient gets unnecessary treatment, but the patient will still survive.

An approach to deal with class imbalance is to use a weighted loss function. This approach is commonly applied in machine learning with success and was also employed in this project lead to much better predictions than when using a unweighted loss function. The reason hereof is that the weighted loss funtion is modified to increase the performance of the minority class. This is referred to an algorithmic approach. Other solutions to the problem of class imbalance is to apply a data approach. This approach is related to resample the data to avoid this issue with imbalanced classes. Commonly over-sampling or under-sampling are employed. In over-sapmling more sample from the minority class are used for training the machine learning algorithms, while in under-sampling the number of samples from the majority class is reduced to get more even

distribution of the classes. The drawback of the former is that often tends to overfit to the training data, while the latter might loose useful information due to the removal of samples from the majority class.

References

[1] "Kaggle 2018 Data Science Bowl," Assessed: June 19, 2019. [Online]. Available: https://data.broadinstitute.org/bbbc/BBBC038/