# Semantic Segmentation of Colorectal Cancer Tissue Images using a U-Net

**Research Project - Advanced Machine Learning SS19**
**Heidelberg University**
**Github:** `https://github.com/flo-he/CRC-Segmentation`

Florian Hess, Valentin Barth

14th October 2019

## Contents

# 1 Abstract

(Valentin Barth)

Automatic segmentation of tumours are likely to improve disease diagnosis, planning and treatment. In the field of medical image segmentation, currently, one of the most popular network architectures is U-Net. In this project we design and assess a U-net implementation for 2D Colorectal Cancer Tissue images using Pytorch.

# 2 Introduction

(Valentin Barth)

The field of medical image segmentation is growing steadily. Especially segmentation of tumours earned a lot of attention in medical science in the last years. Accurate delineation of tumours is necessary for plenty of medical procedures such as monitoring progression or recession. Here, human experts are challenged when distinguishing cancer from healthy tissue manually. This procedure takes a lot of time and is prone to human mistakes. To save both, time and financial means and probably in future to minimise error, it is reasonable to elaborate more and more advanced techniques. The U-net, proposed in 2015 by Olaf Ronneberger et. al., is one of these approaches and lead to very good results in the ISBI cell tracking challenge 2015 and various other challenges. [1]
Now, it is interesting to see how it performs on different data - in this case colorectal Cancer Tissue (CRC) Images. We are also interested in the number of training examples and the extend of data augmentation we need for accurate delineations. We also perform some investigations on the effect of different loss functions on the convergence. Finally, we want to investigate the behaviour of the receptive field when variing the resolution levels, number of channels and other hyperparemeter.

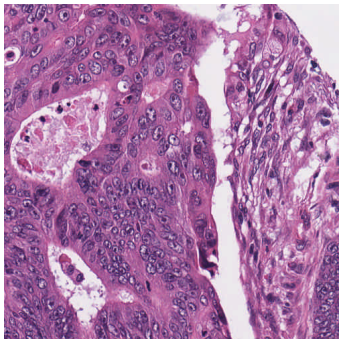
# 3 Exploring the Data

(Florian Hess)

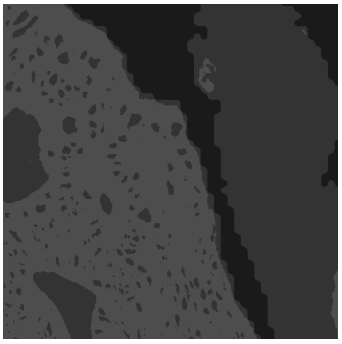
## 3.1 About the Data

Our data set consists of biomedical images of colorectal cancer (CRC) tissue using microscopy imaging. It wraps 20.880 RGB images and grey scale segmentation masks, all of them having an equal size of $500 \times 500$ pixel. We are dealing with a multi-class problem: background (0), tissue (1) and tumourous tissue (2). We will also have to deal with class imbalance, as most images are dominated by background and/or healthy tissue, with tumourous tissue being rather sparse. The segmentation masks are labelled by overlapping the hematoxylin and eosin stain (HE-Stain) images (fig. 1a) with their corresponding Immunohistochemistry (IHC) images (fig. 1b), i.e. the source of the ground

truth segmentation masks is a registration process and not expert labelling. This means we have to deal with the errors that arise from image registration.



(a) An example HE-Stain image from the data set, the background is white while the tissue has a pinkish colour.



(b) Corresponding mask, created by registration of the HE-Stain and Immunohistochemistry image.

Figure 1: A sample from the CRC data set.

## 3.2 Empty Images and Noisy Data Pairs

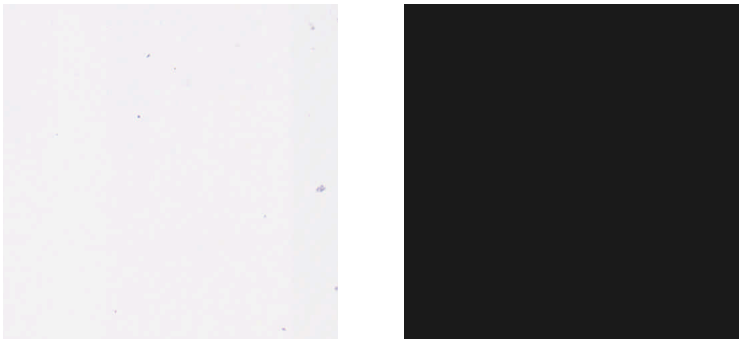
### 3.2.1 Dealing with Sparse Images

Since all the $500 \times 500$ images are only patches of higher resolution microscopy images, there are lots of blank images. However, the corresponding ground truth masks mainly contain class 0, i.e. background (see figures 2a and 2b). Even though this might not hinder training convergence, it definitely slows down training. The idea is that these images contain zero variance and deliver only few information and therefore the corresponding weight updates will not have significant impact on the U-Net's performance. Nevertheless, these images take the same time to be processed on the GPU and may worsen the reliability of batch means.

Hence, we decide to drop images where not at least 1% of the segmentation mask's pixel are of class 1 or 2 (i.e. non background). This technique already thinned out the data set by $\approx 24\%$. However, since we use random patches (see section 5.1.1) while training this problem is not fully solved: A lot of images contain most of their information in a certain corner of the image, so that a randomly cropped patch has a high probability of simply not hitting this area of the image.

### 3.2.2 Dealing with wrongly annotated Data

Due to the registration some of the image-mask pairs do not fit perfectly. This is best shown by some examples: Samples like these may severely hinder training due to simply wrong annotations: They produce unwanted noise in weight updates of the network



(a) HE-Stain image with no tissue apparent.

(b) Mask containing only the background class.

Figure 2: Empty images are a significant part of the data set.

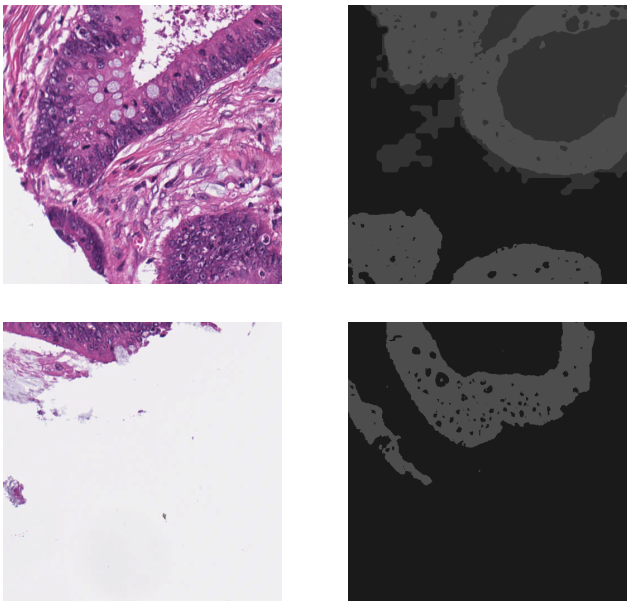


Figure 3: Top row: The HE-Stain image (left) obviously does not show any tissue in the lower left corner, as the mask (right) states. Bottom row: A big part of the image (left) is wrongly classified as tissue (right).

since the network either learns wrong segmentations (depends on the amount of poorly annotated images in the data set), gets punished for not learning wrong segmentations, or does not even learn anything, which is highly unwanted behaviour. Therefore, we try to reduce the amount of these samples as much as possible before the start of the whole training pipeline.

To deal with that, we apply the following algorithm which identifies and drops these

image-mask pairs:

**Algorithm 1** Filter invalid segmentation masks

```
function CheckValidity(img, mask, t, blk_int)
    img ← ToGreyscale(img)                                  ▷ rgb to greyscale via mean
    img ← Invert(img)          ▷ invert intensities, so that background becomes black
    img ← GaussSmoothing(img, σ = 10)                       ▷ fill empty gaps in tissue
    tissue_px ← img[mask == 1 ∨ mask == 2]                  ▷ px annotated as tissue
    empty_px ← tissue_px[tissue_px < blk_int]     ▷ (probably) wrongly annotated px
    n_empty ← empty_px.size
    rel_amount ← n_empty/NonZero(empty_px)                  ▷ rel. amount of wrong px
    if (rel_amount < t) then
        return True                                                  ▷ keep image
    else
        return False                                                 ▷ drop image
```

For each image-mask pair we first convert the RGB image to grey scale and invert the intensities so that background pixel are black rather than white. Due to HE-Stain images having lots of blank spots in the tissue itself, we apply a gauss filter with $\sigma = 10$ blur, to avoid these pixel from being identified as background pixel. We then gather all the pixel that the ground truth mask assumes to be tissue (classes 1 or 2), and compare the pixel positions to the ones in the processed image. With a custom grey scale intensity (*blk_int*) that tells us, up to which intensity we consider a pixel as background, we determine the relative amount of background pixel that are labelled as tissue. With a custom threshold $t$ we then drop the image-mask pair if too many pixel are wrongly classified. By checking cases with bare eye, we found that $blk\_int = 40$ (considering uint8 0-255 range) and $t = 7.5\%$ works fine (e.g. the data in fig. 3 gets dropped by the algorithm). Figure 4 shows good and bad examples of the algorithms selection function.

Applying this algorithm to the data set drops another significant amount of the data so that we are now finally left with $\approx 53\%$ ($\approx 11.000$ image-mask pairs) compared to the initially supplied data. We are in no doubt that there are more sophisticated ways of dealing with this problem, however, we are sure that our approach is at least reasonable.

# 4 The U-Net

(Valentin Barth)

## 4.1 The original architecture [1]

Figure 5 shows the architecture of the original U-net. Basically, it consists of two parts: firstly the contracting part where it gains context information (encoder) and secondly the expanding part localising this information (decoder).

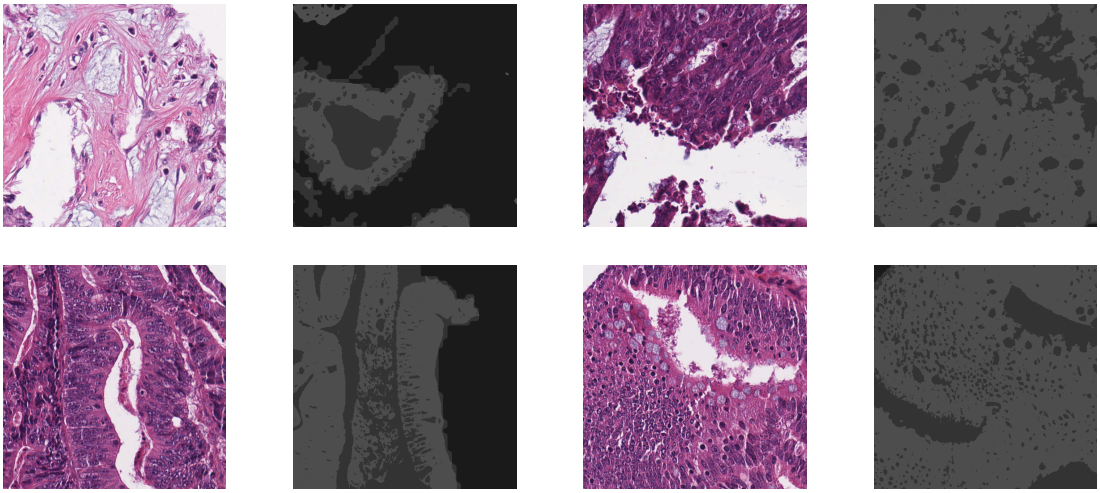


Figure 4: Data dropped by the proposed algorithm. Top row: Cases where the algorithm correctly dropped the images. The masks annotate background pixel in the middle of image as cancer tissue. Bottom row: Cases where the algorithm failed. Even though significant parts of the image are annotated wrongly, they still pass the selection.

#### 4.1.1 Encoder

The input is a grey-scale image (1 channel) of size $572^2$. The contracting part consists of 4 similar blocks. Each of them performs two convolutions with a $3\times3$ kernel (always with activation via ReLU) followed by a max-pooling operation with both, Kernel size and stride 2. Since the convolution is performed without padding, the resolution shrinks by two each time. This is due to the following relation: Say the input resolution is $n\times n$ and the kernel size is $k\times k$ than the output resolution is $(n-k+1)\times(n-k+1)$. Furthermore, the output is halved each time a max-pool operation is applied. At the same time, within each block the first convolution doubles the number of channels (except for the first layer leaping from 1 to 64). Through this procedure, we increase the context information but we lose certainty in localisation. The encoder path is finished at a resolution of $32^2$ pixel and a channel size of 1024.

#### 4.1.2 Decoder

The expanding path is based on the a 4-block structure as well. However, there are two crucial differences. The max-pool operation is replaced by a $(2\times2)$ up-convolution doubling the resolution and halving the number of feature maps. Additionally, to get a higher local confidence, the respective layers of the contracting path are cropped to the right size and concatenated with the up-conv. layers (apparently, this doubles the number of feature maps again). Now in the first convolution of each block we do not double the number of feature maps, but we halve them. After processing the 4 blocks

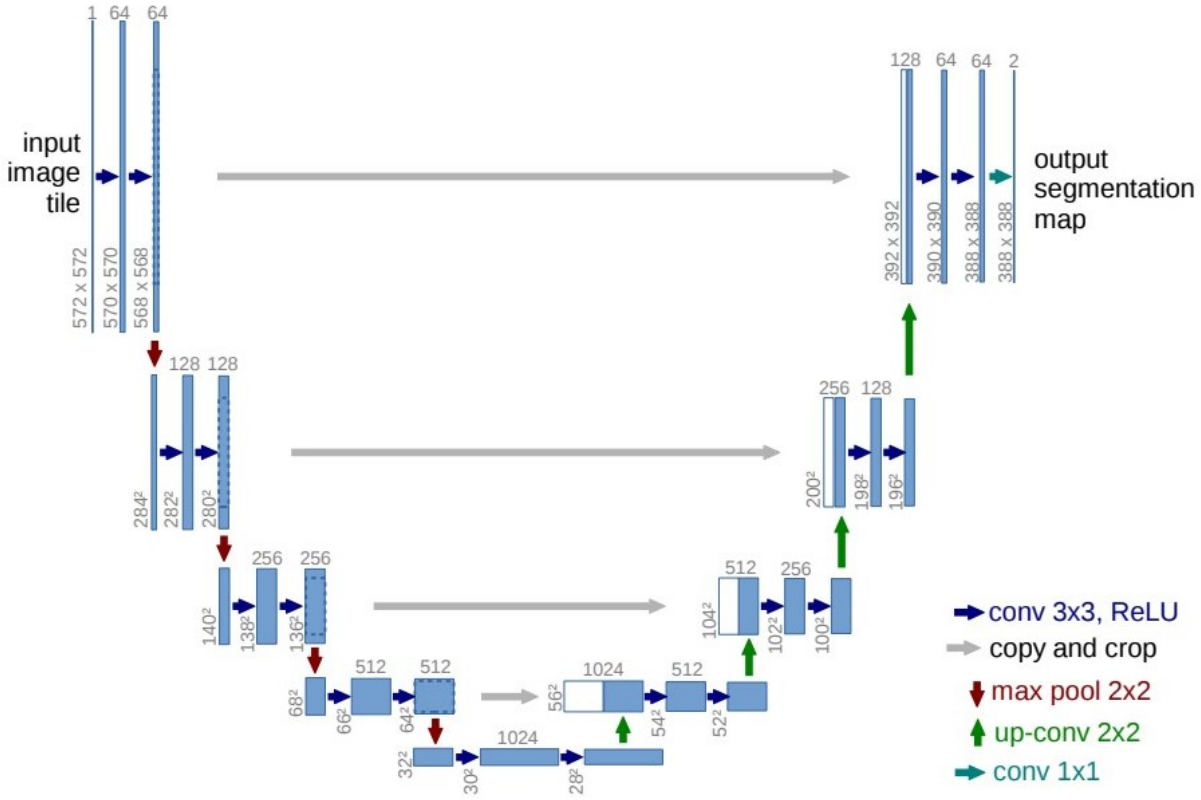



Figure 5: Original U-net architecture. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. [1]

a $(1 \times 1)$ convolution is applied yielding the output segmentation map with one channel per class and a resolution of $388 \times 388$ pixel.

## 4.2 Modifications in our implementation

However, we decided to do some small changes:

1. Since we deal with rgb images, we have 3 input channels instead of 1.
2. Due to our restrictions in computational power we reduced our input to $256 \times 256$ patches of our original images (explained in more detail in section "training process") and increased the number of channels only to 32 (instead of 64) in the first convolution. This leads to of final number of channels of 512 (instead of 1024). These sanctions might reduce the accuracy of our neural net but they are necessary to make training feasible with our means.
3. The original version uses no padding. However, we decided to enable padding

for three reasons. Firstly, with a padding of 1 pixel we inhibit shrinking of the resolution which has the pleasant effect that our net is completely symmetric and we do not have to crop the layers before concatenating them to the expanding path. While this is mostly for convenient reasons, the second effect is that, during testing, the output images have the same size as the input images this way. Moreover, we prevent a loss of information at the rim and in the corners this way (a "natural" drawback of the pure convolution).

4. Finally we decided to enhance the convolution blocks with batch normalisation before each activation to avoid/reduce the problem of vanishing or exploding gradients. Additionally, we swap ReLU activations for LReLUs (Leaky ReLUs). Both of these changes are adopted by [2].

# 5 Training Process

(Florian Hess)

## 5.1 Training Strategies

### 5.1.1 Data Preprocessing and Augmentation

Before an image of the data set hits the input layer of the U-Net, a few steps take place: Since the U-Net is quite large, we are limited by GPU VRAM (6GB). The original paper [1] also had only 6GB available and coped with that by simply using a batch size of 1 with a high SGD momentum, so that the weight updates are heavily based on previously seen images. However, to train with a reasonable batch size, we use Random Crops to crop the initially $500 \times 500$ sized images and masks to $256 \times 256$ patches. This has the additional benefit of data augmentation. Additionally, dealing with powers of two eases the architecture of the U-Net, since the pooling and up-convolution operations halve and double image sizes. We then use Random Flips, both horizontally and vertically, for even further data augmentation. Finally, the image patch gets normalised by the average means and standard deviation of the whole training set per RGB channel.

[1] also used elastic deformations of the images which seemed to be a key factor in their success of training the U-Net on their data set. However, they had to cope with very few data ($\approx 30$ training images). In our case, with thousands of training images available, we are confident that our data augmentation steps do just fine.

### 5.1.2 Cross Validation

To apply model selection and avoid overfitting without touching the U-Net architecture or use further data augmentation, we use training based on cross validation: In each epoch we apply $k$-Fold cross validation, i.e. each epoch holds $k$ slightly differently trained models each trained on $(k-1)$-folds and validated on the leftover fold $k$. We then keep the weights of the model that generalises best, i.e. yields the lowest validation loss, which becomes the starting point for the following epoch.

### 5.1.3 Learning Rate Scheduling

Reducing the learning rate when the model struggles to further reduce the loss objective during training is an useful way to help the model dive deeper into its currently found minimum. There are several ways to determine when to reduce the learning rate and by how much. Reducing the learning rate too early might stop the model from finding a better minimum, so we decide to reduce the learning rate when validation loss plateaus for several epochs of training. This allows the model to find good local minima in the beginning of training (high learning rate) and to steadily fine tune its weights towards the end of training (decreasing learning rate). This functionality is already implemented in Pytorch via the ReduceLRonPlateau class.

For further inside on the training progress and a better measure of when the loss plateaus we keep exponential moving averages of the training and validation loss:

$$x_{EMA} = \begin{cases} x_e & \text{if} \quad e = 1, \\ (1-\alpha)x_e + \alpha x_{EMA} & \text{else} \end{cases} \tag{1}$$

The the label x describes the training or validation loss, $\alpha$ is an weighting parameter and $e$ is the current epoch. Reducing the learning rate is then based on the EMA of the training loss, rather then the training loss of the current epoch itself. We use $\alpha = 0.9$ throughout training.

## 5.2 Final Training Setup

For training we split the filtered data set set using 80% for training and the remaining 20% for testing. For each experiment we train the model for a maximum of 250 epochs. One epoch is defined as processing 250 batches. We use 5-fold cross validation, i.e. per epoch the model trains on 200 batches and is validated on 50. This equals a total maximum of 50.000 weight updates. Using a batch size of 16 means that the U-Net sees $16 \times 250 = 4000$ images of the available 8827 training images per epoch. Data augmentation is applied on the fly. We use the long proven Adam optimiser with a initial learning rate of $5 \cdot 10^{-4}$ and a weight decay of $3 \cdot 10^{-5}$. We drop the learning rate by a factor of 0.2 if the exponentially moving average of the training loss is not improving for 5 epochs. The minimum learning rate is $10^{-6}$ and training is interrupted if the validation loss does not decrease for several epochs or the maximum of 250 epochs is reached. All models are trained on an NVIDIA GeForce GTX 1060 6GB GPU.

# 6 Experiments

## 6.1 Investing the effect of different Loss functions

(Valentin Barth)

One part of our experiment will be to test the effect of different loss functions on training and performance (in terms of evaluation scores). We test the following loss functions:

### 6.1.1 Categorical Cross-Entropy (CCE)

The go-to loss for image segmentation is the pixel-wise (categorical) cross-entropy loss:

$$CE = \frac{1}{|B|} \sum_{b \in B} \frac{1}{|I|} \sum_{i \in I} \sum_{c \in C} t_{i,c}^{b} \log(p_{i,c}^{b}), \tag{2}$$

where $B$ is the set of batch elements, $I$ is the set of all pixel of a given prediction mask and $C$ is the set of classes. $t_{i,c}^{b}$ is the ground truth per batch instance per pixel and is one hot encoded, i.e. it is 1 for the correct class $c$ and else 0. $p_{i,c}^{b}$ is the softmax output per batch instance per pixel of class $c$ under consideration. I.e. cross entropy gets averaged over all pixels and the batch. Due to its smooth gradients, the CE loss provides stable training. However, each pixel is weighted the same, which becomes a problem for heavily imbalanced data sets. To compensate, re-weighted cross-entropy applies weights to each class in (2):

$$WCE = \frac{1}{|B|} \sum_{b \in B} \frac{1}{|I|} \sum_{i \in I} \sum_{c \in C} w_c \cdot t_{i,c}^{b} \log(p_{i,c}^{b}) \tag{3}$$

In our case, we deal with the following approximate mean relative class occurrences per instance:

- class 0 background $\approx 52.5\%$
- class 1 tissue $\approx 27.5\%$
- class 2 tumourous tissue $\approx 20.0\%$

To increase the effect of the rare classes' loss terms, we design the weights to be inversely proportional to the class occurrences. We compute the class weights as follows:

$$w_c = \frac{1}{r_c \cdot \sum_{i \in C} \frac{1}{r_i}}, \tag{4}$$

where the $w_c$ is the class weight and $r_c$ is the relative class occurrence of class $c$. (4) ensures that the weights are normalised, i.e. sum up to 1 instead of scaling the terms with $w_c > 1$, which would result in different behaviour. Computing the $w_c$ gives us:

- $w_0 \approx 0.181$
- $w_1 \approx 0.345$
- $w_2 \approx 0.474$

### 6.1.2 Sørensen–Dice Coefficent (DSC)

Besides CE a common choice for a loss function, especially in medical image segmentation, is the Dice Similarity Coefficient (DSC). It measures the overlap of two sets $A$ and $B$ by computing:

$$DSC = \frac{2 \cdot |A| \cap |B|}{|A| + |B|}, \tag{5}$$

i.e. it measures the intersection of two sets in respect to the sum of elements of both sets. Since (5) is not applicable for training usage due to not being differentiable, there are several formulations of a trainable loss for the DSC. We use the formulation used by [2]:

$$DICE = -\frac{2}{|B|} \sum_{b \in B} \frac{1}{|C|} \sum_{c \in C} \frac{\sum\limits_{i \in I} t^b_{i,c} \cdot p^b_{i,c}}{\sum\limits_{i \in I} t^b_{i,c} + \sum\limits_{i \in I} p^b_{i,c}}, \tag{6}$$

where $t^b$ is a one-hot encoded ground truth mask and $p^b$ again are the softmax values of the output of the U-Net, i.e. both are 3-dimensional tensors of shape $[C, H, W]$.

## 6.2 Comparing two U-Nets of different depth

(Valentin Barth)

### 6.2.1 Setup

To keep the model small for further results, we test if we can use a smaller U-Net than the one used by [1]. I.e. We drop the lowest resolution level and halve all feature maps, leaving the network with 4 levels where each level has 32, 64, 128 and 256 feature maps respectively. To avoid too severe underfitting, we do not use dropout with this model. For comparison we train a larger network, i.e. we add a level with 512 feature maps which becomes the bottleneck. We add dropout with $p = 0.5$ after every convolution but the $1 \times 1$ convolution producing the output segmentation map to counteract overfitting. Both models are trained on a combination of the cross-entropy loss (2) and Dice (6):

$$\mathcal{L} = CE + DICE \tag{7}$$

The models are tested using per pixel accuracy and Dice score, where the former is

$$ACC = \frac{\sum_{i \in (P \cap T)}}{\sum_{i \in T}}, \tag{8}$$

where $P$ and $T$ are the predicted and ground truth segmentation masks, i.e. $i$ runs over the pixel of the images. The latter is simply the negated Dice loss (6):

$$DICE_{score} = -DICE \tag{9}$$

While testing, we feed the network the full sized $500 \times 500$ images and symmetrically mirror them to $512 \times 512$. This has two reasons: First, due to our architecture the shape of the input images has to be in powers of two (consider $500 \times 500$ images: after the third level, the pooling would have problems halving the size of $125 \times 125$). Secondly, supplying extra border pixel feeds the network with more context for prediction of the border pixel of the output mask. [1] used this technique throughout training, too. Due to our symmetric U-Net, we crop the output segmentation maps back to $500 \times 500$ to align with the ground truth masks. Even though we train with $256 \times 256$ patches, the performance of the U-Net on the full images is worsened due to the nature of convolutional neural networks and their weight sharing (i.e. the kernel weights of the convolutional layers are the same for all positions the kernel is applied on in the input image). We test on a total of 2206 images. Test metrics are computed with the model state from the epoch where the validation loss was smallest. Since we saved the progress in intervals of 5 epochs, we take the model closest to the saved model state.

### 6.2.2 Results

The two models are compared in the following table:

| | Epochs | best Epoch | t/Epoch | DICE | ACC |
|---|---|---|---|---|---|
| Small U-Net | 85 | $\approx 85$ | $\approx 450s$ | 0.4394 | 0.7541 |
| Large U-Net | 60 | $\approx 60$ | $\approx 530s$ | **0.4618** | **0.7763** |

Table 1: Comparison of the two models.

Table 1 shows that the larger, pruned model outperforms the smaller model in respect to the chosen test metrics. Looking at the loss curves (see figure 6) shows, that the pruned model handles generalisation better throughout training thanks to the additional dropout layers. However, using dropout (especially with high drop probabilities) on the smaller model would prune it to an extend where it would rather lead to underfitting. It is also interesting to see that the larger model took less epochs to converge even though using dropout generally increases training time, since the network has to handle missing input signals and get rid of co-adaptation effects.

However, taking a look at the metric values itself one would say that both networks perform poorly. A huge problem is the noisy data set with its not that uncommon samples where the ground truth masks are rather wrong. Since it would be interesting to see how the networks handles such cases, we add some predicted segmentation masks to interpret the models performance visually.

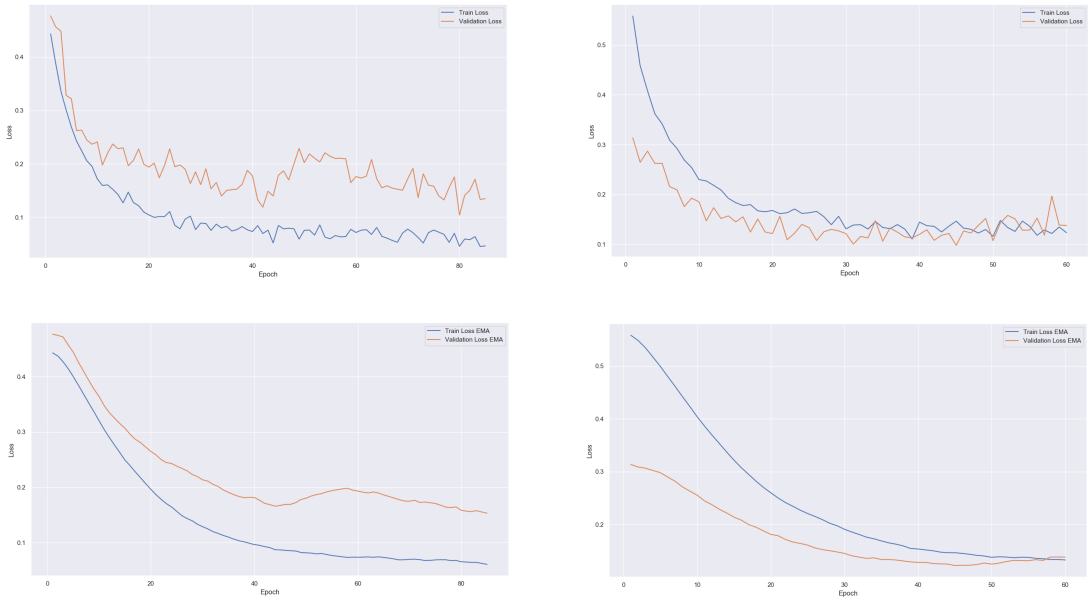
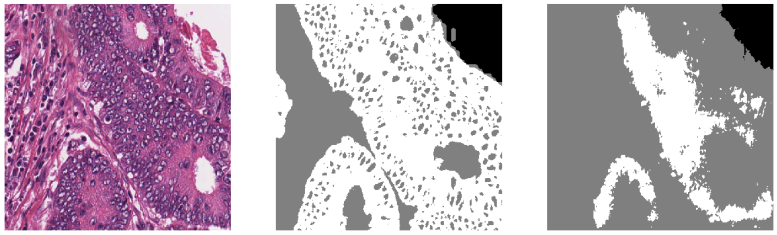
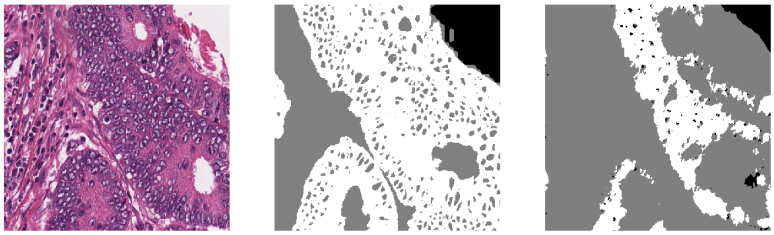



Figure 6: Loss curves of both models. (Left column) Small model. (Right column) Large and pruned model. The larger, pruned model has an overall smaller validation loss.



(a) Small U-Net: The network does not fully capture the tumourous tissue let alone the fine details of normal tissue between the tumourous tissue. It learns to smoothen the white spots in the input image and classifies them as tissue.



(b) Large U-Net: Compared to the predictions of the smaller network, the larger one classifies some the small gaps in the tissue as background. Even though this is in theory right, the ground truth masks are generally all smoothed and state that the gaps are tissue, too.

(c) Small U-Net: The model struggles to classify the tumourous tissue and only predicts healthy tissue in the tissue area (it is all white since there are only classes 0 and 1 present and the plotting adjusted to the grey-scale range).

(d) Large U-Net: Here the net smooths out the gaps in the tissue and predicts some tumourous tissue, which, however, does not really align with the mask.

Figure 7: Example predictions of the networks. (left) Input images, (middle) ground truth masks, (right) predictions.

Analysing figure 7 reveals two problems. Consider 7c and 7d: First, tumourous tissue is generally rather blue/purple which can be clearly seen in the images. The ground truth mask's prediction of tumourous tissue is however not really aligned with the input image. Second, a significant part of the ground truth mask is background even though there is clearly tissue in the input image (sadly, we have only dealt with the opposite case while filtering the data set). However, the predictions do a good job in predicting the background/tissue boundaries. This is a problem, since the wrong ground truth masks penalise the network for this behaviour, even though the network is right. This also explains the rather disappointing results of the test metrics in table 1 and raises the question, if there is much room for improvement without having to filter the data set even more.

## 6.3 Training on different Loss functions

(Florian Hess)

In this section, we will investigate the effect of different loss functions in training our model.

### 6.3.1 Setup

Our baseline architecture of the U-Net will be the same as the large model of section 6.2, i.e. 5 resolution levels, with the deepest featuring 512 feature maps, dropout with $p = 0.5$ after every convolution except for the output block and batch normalisation. We train 4 different networks, each using a different loss criterion: cross-entropy (2), re-weighted cross-entropy (3), Dice (6) and a combination of cross-entropy and Dice (7). For WCE we use the weights computed in section 6.1.1. Again, test metrics are Dice score and per pixel accuracy. We will also consider the loss curves and some example segmentations as a matter of analysing the different models' performances.

### 6.3.2 Results

The results of the test metrics can be seen in table 2: Let's concentrate on the winning

| Loss | Epochs | best Epoch | t/Epoch | DICE | ACC |
|---|---|---|---|---|---|
| CE | 130 | $\approx 50$ | $\approx 475s$ | 0.4540 | 0.7692 |
| WCE | 105 | $\approx 30$ | $\approx 475s$ | 0.4459 | **0.7766** |
| DICE | 175 | $\approx 175$ | $\approx 517s$ | **0.5046** | 0.7506 |
| DICE + CE | 60 | $\approx 60$ | $\approx 530s$ | 0.4618 | 0.7763 |

Table 2: Comparison of different loss functions.

models regarding Dice score and per pixel accuracy: While the model trained on pure Dice loss achieves by far the best Dice score, it also yields the lowest pp-accuracy. Vice versa, the model trained on weighted cross entropy scores the highest pp-accuracy and

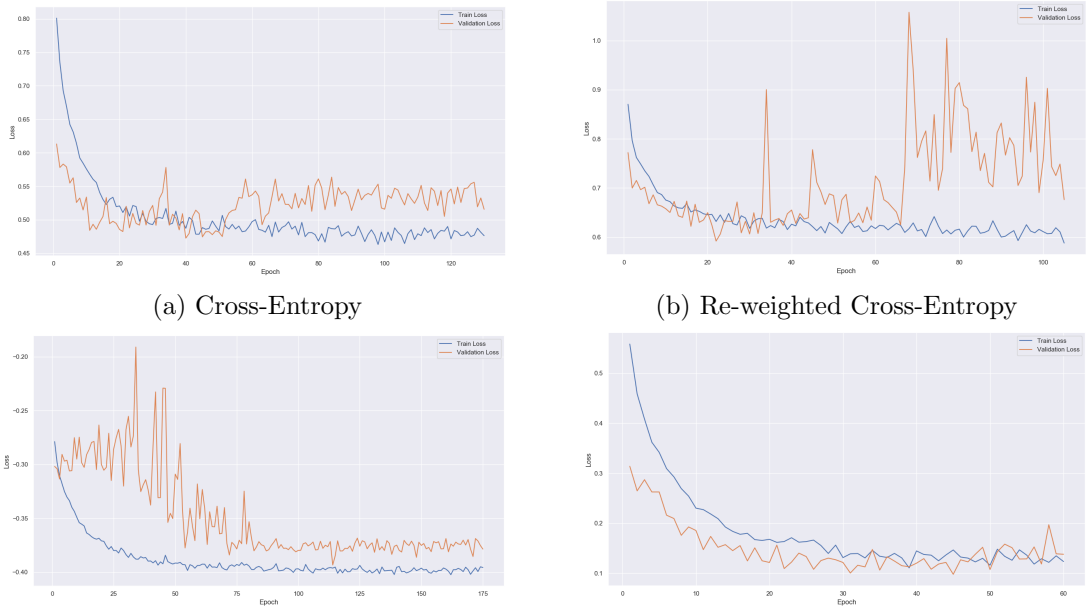
the lowest Dice score. The former was expected, since the main objective of training was reducing the Dice loss and hence maximising the Dice score. It remains to see if the reason for the metrics playing against each other can be found in some example predictions of the networks. It feels natural that the combination of the loss objectives balances the test metrics: The model trained on both Dice and cross-entropy is very close to the best pp-accuracy achieved, while still being second place in regards to the Dice score. The models trained on Dice took generally longer per epoch, which is due to more computationally expensive gradients of the Dice loss: CE has very elegant gradients w.r.t. the logits $s_i$:

$$\frac{\partial CE}{\partial s_i} \propto p_i(s_i) - t_i, \tag{10}$$

where, $p_i$ are the softmax outputs of the logits $s_i$ and $t_i$ the ground truth labels. Dice, however, is a tad more complex:

$$\frac{\partial DICE}{\partial s} = \frac{\partial DICE}{\partial p} \cdot \frac{\partial p}{\partial s} \propto \frac{2t^2}{(p+t)^2} \cdot \frac{\partial p}{\partial s}, \tag{11}$$

where we dropped the index notation for simplicity and $\frac{\partial p}{\partial s}$ is the derivate of the softmax w.r.t. the logits. Computing the Dice loss and the backwards pass is therefore more expensive and the denominator of (11) holds the risk of making the gradients blow up if $p + t \approx 0$. This instability can be seen in the training curves of these models. Taking




(a) Cross-Entropy

(b) Re-weighted Cross-Entropy

(c) Dice

(d) Dice + Cross-Entropy

Figure 8: Loss curves of the models from table 2.

a look at the model trained on pure Dice (subfig. 8c), one can clearly see the validation loss going all over the place in the first third of training, which might be due to the

mentioned risk of gradients blowing up. Especially since the combination of CE and Dice (subfig. 8d) yields a very smooth training curve. This problem could have been solved by simply adding a small number $\epsilon$ to the denominator in (6), which we sadly did not use. The model trained on Dice is also the only one that constantly yields a higher validation than training loss, which might indicate that a model trained on Dice needs more pruning. Even though all of the models converge after several epochs, lowering the learning rate, as mentioned in 5.2, always led to the models overfitting (as can be seen in subfigures 8a, 8b and 8d). Considering WCE (subfig. 8b), we can not explain the heavy oscillations in the validation loss which increase over the course of training. However, considering the results in table 2, WCE did converge faster than plain CE.

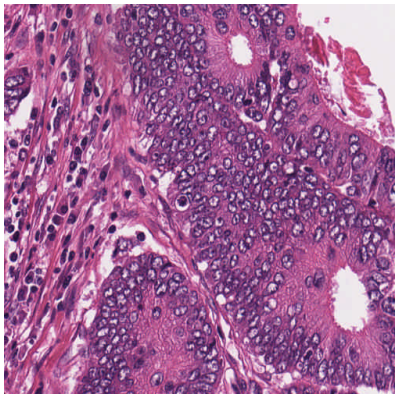
### 6.3.3 Interpreting the Models' Predictions

As in 6.2.2, we want to compare the models' performances visually by looking at their predictions of some test samples (see figures 9, 10 and 11). The top row in all of the figures are the input image and the ground truth masks, while the rest are predictions made by the different models of 2. The difference in gray-scale values between the ground truth masks and the predictions is due to different scaling while plotting and saving the predictions, which does not change the distribution of classes, i.e. from darkest to brightest the classes are background, tissue and tumourous tissue.

Let's first consider figure 9. First, only the model trained on WCE applies smoothing to the background spots in the tissue, i.e. predicts tissue like the ground truth masks generally do. This makes sense, since the background class is weighted less and therefore the model does not really focus on extracting the information that practically all blank spots in the input image are of the background class. This is confirmed by the model trained on pure CE: Here the model (kind of) predicts most of the obvious blank spots as background. One can observe that the weighting of the classes in WCE really changed the models' predictions: The model receives a higher penalty for predicting tumourous tissue in wrong places, which explains why the WCE predictions is lacking the fine-grained segmentations of the tumours on the left of the image, as they are present in the pure CE counterpart. What is really interesting, is that the CE model and the Dice model both heavily predict tumourous tissue in spots that do not align with the ground truth. However, looking at the image the spots in the left half of the image do look like tumourous tissue (blue/purple cells). The detailed segmentations of the model trained on Dice are interesting, too: It really tries to nail all the fine-grained borders of healthy tissue in between the tumourous tissue and all the blank spots in between them. The model trained on both Dice and CE, however, disappoints in the sense that it is intuitively/subjectively the worst prediction. It does not look near any of the models' predictions of the pure counterparts (9e, 9c).

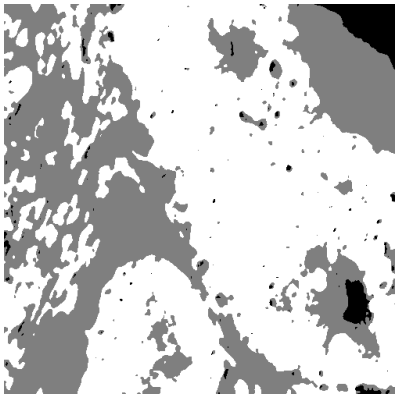
Figure 10 shows an example of a that comes with a poorly annotated ground truth mask. CE yields the most convincing prediction, while WCE and Dice+CE are rather disappointing. The on Dice trained model yields similar results to 9e: Detailed segmentations but also very generous in predicting tumourous tissue. In figure 11, CE and Dice both yield pleasing results, however, Dice is the clear winner, as the pp-accuracy is

practically identical while the difference in Dice score is quite high.



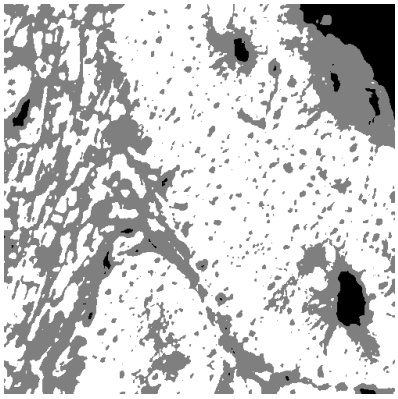
(a) Input Image (not mirrored)

(b) Ground-truth mask

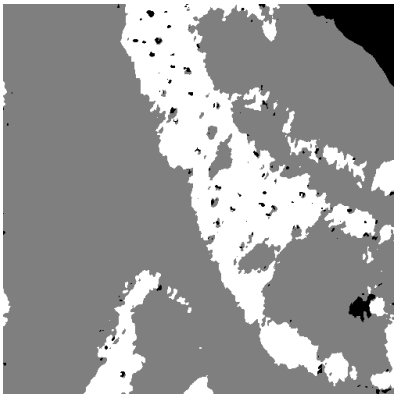


(c) CE: $DICE = 0.4000$, $ACC = 0.6241$

(d) WCE: $DICE = 0.4376$, $ACC = 0.6283$

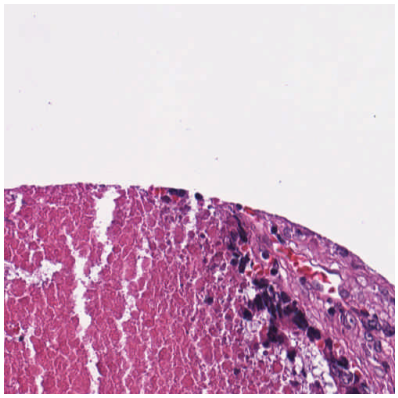


(e) Dice: $DICE = 0.5667$, $ACC = 0.6138$



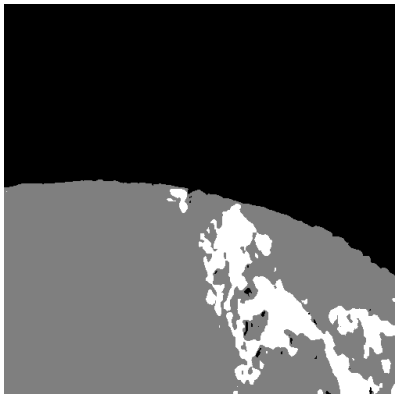
(f) Dice+CE: $DICE = 0.4164$, $ACC = 0.5152$

Figure 9: Comparison between the models of a sample from the test set.

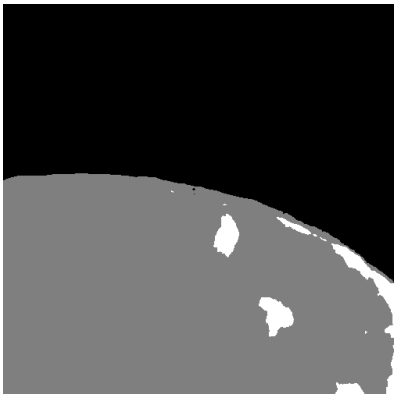


(a) Input Image (not mirrored)

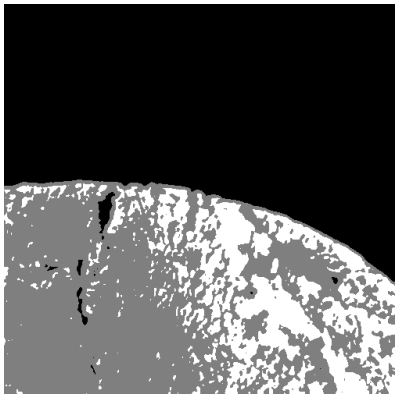
(b) Ground-truth mask



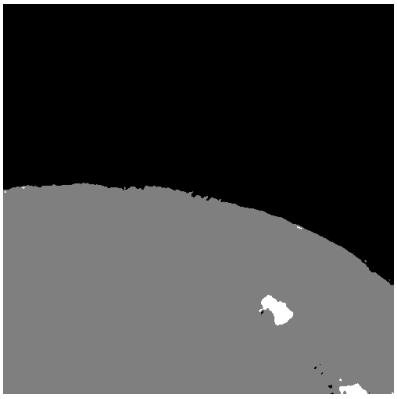
(c) CE: $DICE = 0.4553$, $ACC = 0.7132$



(d) WCE: $DICE = 0.4380$, $ACC = 0.7131$

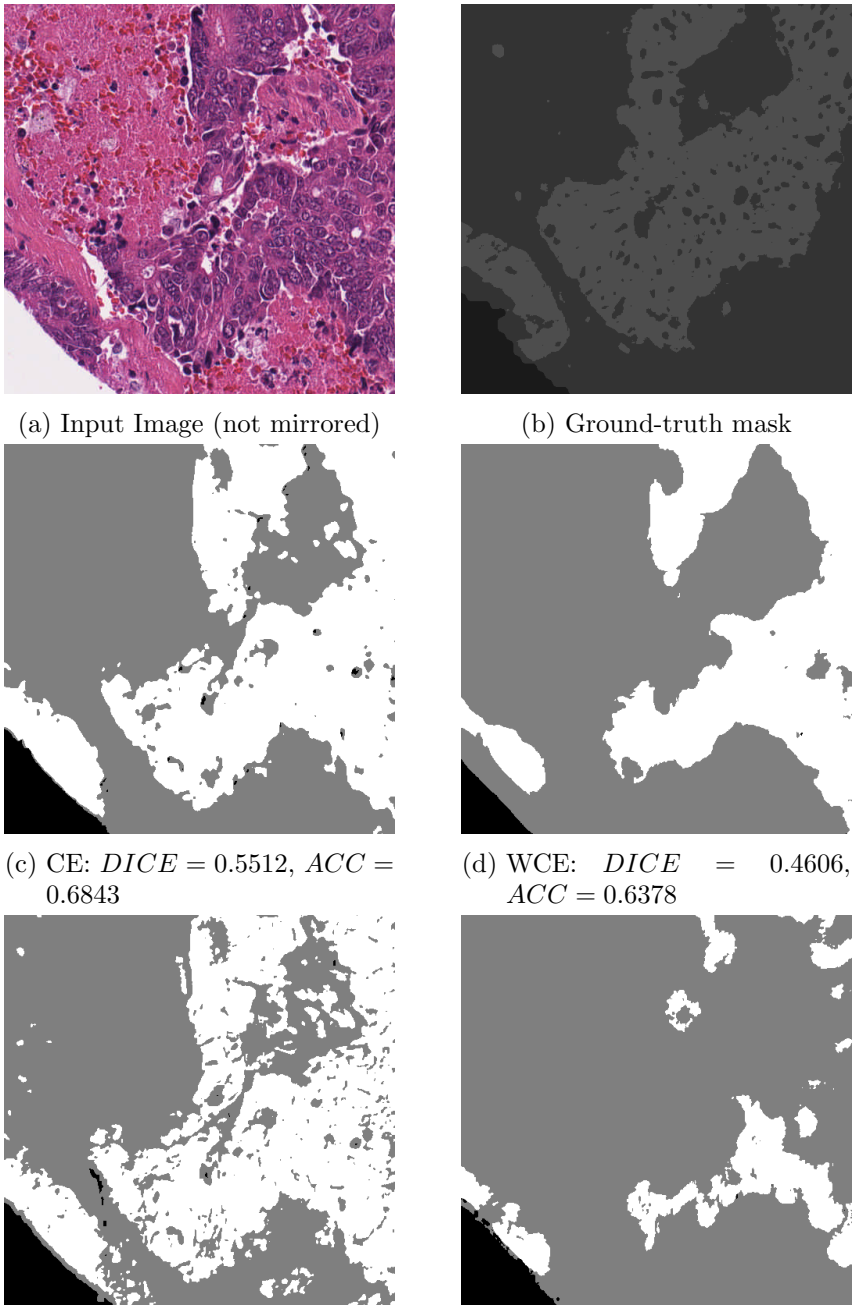


(e) Dice: $DICE = 0.5101$, $ACC = 0.7010$



(f) Dice+CE: $DICE = 0.4670$, $ACC = 0.7357$

Figure 10: Comparison between the models of a sample from the test set.



(a) Input Image (not mirrored)

(b) Ground-truth mask

(c) CE: $DICE = 0.5512$, $ACC = 0.6843$

(d) WCE: $DICE = 0.4606$, $ACC = 0.6378$

(e) Dice: $DICE = 0.7494$, $ACC = 0.6819$

(f) Dice+CE: $DICE = 0.5100$, $ACC = 0.6020$

Figure 11: Comparison between the models of a sample from the test set.

The reason for the model trained on the combination of Dice and CE predicting more like the CE model than the Dice model may be due to the components of the loss (see (7)) having different value ranges. While Dice is in the range of $[0, 1]$, CE is not bounded

due to $CE \propto \log(p)$ with $p \in [0, 1]$. This means that the Dice contribution is lower than the CE contribution by design. A solution to this would be to use the log of dice, to balance the contributions:

$$\mathcal{L}_{balanced} = \log(-DICE) + CE, \tag{12}$$

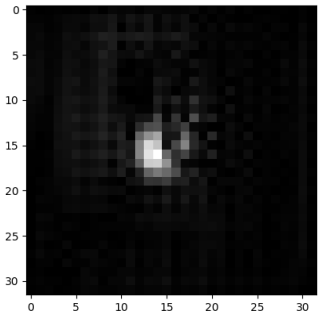
where DICE $\in [-1, 0]$ due to our definition as a loss in (6). However, we sadly did not have any time left to train a model with (12).

## 6.4 Receptive Fields

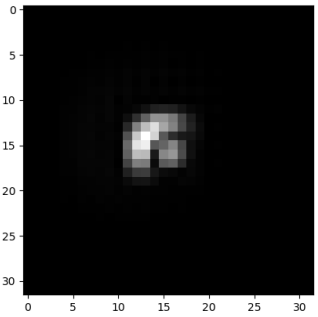
(Valentin Barth)

The Receptive Field (RF) of a convolutional network denotes how many pixel of the input affect one pixel of the output. We distinguish between the theoretical and the effective RF. While theoretically the RF is increased by a one pixel border for every convolution we used here, is doubled for every pooling and has a quadratic shape, the effective RF is generally way smaller and has a gaussian-like intensity distribution. To calculate an approximation of the theoretical RF and to visualise the effective RF, we create a zero tensor and feed it into the network. From the output we select one pixel and calculate the gradient with respect to the input tensor using pytorchs autograd functionality. For the theoretical RF, we determine all pixel intensities which are non-zero. This means already an approximation, since gradients that get rounded to zero by the autograd.grad function (due to the value range of 32-bit floats) are not taken into account. The effective RF can be delineated by bare eye. Figure 12 exemplarily shows the effective receptive fields of some of our models, denoted with the theoretical RF size.

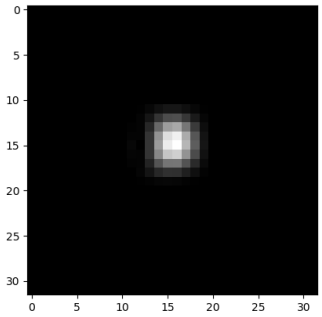
One can see that our way of determining the theoretical RF is just an approximation, since the Dice model and the Dice+CE model both have the same architecture, yet different theoretical RFs. The Dice model apparently learned to even further ignore more distant pixel to the pixel under consideration, so that the gradients in the outer square get rounded to be zero (therefore the difference $172 \times 172$ to $116 \times 116$ for the dice model). There is a huge difference between the theoretical and the effective values. We suppose that this is due mainly to the following reason. In a convolution the pixels at the margin are not as crucial as central pixels. This effect applies at every convolution leading to very low influence of the outer pixels, hence, for gradients close to zero. Actually, they are so small that the intensities on the images are not noticeable by eye. This is why we see extremely narrow effective RFs in fig 12. However, we do not really have an explanation for the different shapes in relation to the different loss functions and as far as we know this is still subject of current research.




(a) RF of the small U-Net with only 4 resolution levels (Dice+CE). Theoretical RF: (84 × 84)




(b) RF of corresponding full U-Net (Dice+CE). Theoretical RF: (172×172)




(c) RF of the full U-Net trained on pure Dice-loss. Theoretical RF: (116 × 116)

Figure 12: Receptive Fields of different models. The brighter the pixel, the higher the gradient magnitude.

# 7 Conclusion

(Valentin Barth)

In conclusion, we found that the results of our experiments (especially the loss comparison in section 6.3 and the computation of the receptive field in 6.4) yielded very interesting results. However, we underestimated the effect of poorly annotated data, i.e. in the verdict we should have spent more time on taking care of the data set in form of further filtering and perhaps using more data augmentation to compensate the resulting lack of data. Of course expert labelling would be tremendously useful, however, in practice such data is rare and therefore on has to come up with ways to make use of "noisy" data sets at hand. This tackles another problem we had, namely that our chosen test metrics do not really tell a lot about the U-Nets' performances on the data set. Test metrics that take spatial shifts of segmentations in the masks into account might be useful, e.g. the Hausdorff-Distance as used in medical image segmentation [3]. It would have also been interesting to use the balanced loss (12) to train another Net and compare it to the rest.

Considering our U-Nets' predictions, it would be very interesting to hear an expert's opinion on the produced segmentations for images that have an unfitting ground truth mask (e.g. cases like fig. 10 or even 11). If one of our network produces reasonably well segmentation masks, they could be used as new "ground truth masks" for all the poorly annotated samples we dropped earlier on. This could, for example, help to train another model to further boost its performance on the data set.

# 8 Acknowledgements

A special thanks to Dr. med. Weis and his group who supplied us with the huge data set with over 20,000 crc images and corresponding masks! Moreover thanks to Prof. Dr. Hesser for arranging contact. We also want to thank Lynton Ardizzone for helping us out with problems throughout the project.

## References


[1] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[2] Fabian Isensee, Jens Petersen, André Klein, David Zimmerer, Paul F. Jaeger, Simon Kohl, Jakob Wasserthal, Gregor Koehler, Tobias Norajitra, Sebastian J. Wirkert, and Klaus H. Maier-Hein. nnu-net: Self-adapting framework for u-net-based medical image segmentation. *CoRR*, abs/1809.10486, 2018.

[3] Davood Karimi and Septimiu E. Salcudean. Reducing the Hausdorff Distance in Medical Image Segmentation with Convolutional Neural Networks. *arXiv e-prints*, page arXiv:1904.10030, Apr 2019.