NYILATKOZAT

Név: Bakos Bence

ELTE Természettudományi Kar, szak: Alkalmazott matematikus MSc

NEPTUN azonosító: LMI9I7

Szakdolgozat címe:
Medical image segmentation with deep learning

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2021.05.30.

[illegible]

a hallgató aláírása

Eötvös Loránd University
Faculty of Science
Institute of Mathematics

Medical image segmentation with deep learning

Applied mathematics MSc thesis

Bence Bakos

Supervisor:
András Lukács

Budapest
2022

Contents

1 Introduction

It is undeniable, that AI and deep learning has revolutionized and will continue to revolutionize many areas of our everyday life. We use these technologies every day already and the progress is not slowing down. Every year we see new models come up with better performance on benchmark datasets than the ones in current use. However applying these models in practice often brings up unexpected obstacles.

Computer vision is a great example of this. We have seen during the years the evolution of classification networks and their performance on benchmark datasets like CIFAR or ImageNet. But when these models are applied for a specific usecase, adjustments have to be made.

The topic of my thesis is a specific task in computer vision, called semantic segmentation. The goal in this task is to specify the location of the different types of objects on an image. This area is intensively researched due to its importance for example in the field of self-driving cars and medical image analysis. My goal is to give a broad introduction to semantic segmentation in deep learning and demonstrate it on medical datasets. For this I have to go through the task specific deep learning models as well as the domain specific obstacles which can come up when working with different medical image datasets.

In the Section 2 I will define the image domain and lay down some basic notion in computer vision. Then the basics of some computer vision tasks will be introduced which will lead to semantic segmentation. In Section 3 I dive deeper into semantic segmentation. I will define the task formally and introduce some applications of it alongside with some common challenges in the area. Then in Section 4 I will give an overview of popular segmentation models emphasizing one particular model family in medical imaging, the U-Net. Following that, in Section 5 I will cover loss functions and metrics for the task. These topics will be investigated in the light of the common obstacles in semantic segmentation and medical imaging. In the final section (Section 6) I will give some experimental result about the aforementioned topics. I am going to introduce some medical image datasets, describe the details of my experiments on them and present my results.

Required knowledge

In this thesis I am going to assume that the reader is familiar with the basics of deep learning. These basics include the notion of MLP (multi-layer perceptron) networks, definition of a loss function and gradient descent with backpropagation. I will also rely on convolutional neural networks. In this field I assume the understanding of convolutional layers and how a standard convolutional network like VGG-16 is constructed. Beside these I might refer to some advanced, but nowadays fairly standard techniques like batch normalization and residual networks.

2 Computer vision

2.1 Image domain

In deep learning one of the largest and most advanced area is computer vision. The domain here contains images (or sequences of images in case of videos). If we want to mathematically formalize a task in computer vision, we have to clarify how we interpret images as a mathematical concept.

We can think of images as tensors with dimensions $H \times W \times Ch$, where $H, W, Ch \in \mathbb{N}$. Here H and W are the height and width of the image. For an image $I \in \mathbb{R}^{H \times W \times Ch}$ and spatial coordinates $1 \leq i \leq H, 1 \leq j \leq W$, the (i, j) pair defines a *pixel* of the image. The values of a pixel is given by the vector $I[i, j] \in \mathbb{R}^{Ch}$, where Ch is the number of channels. For example a black and white image has one channel and has a form of $I \in \mathbb{R}^{H \times W \times 1} (= \mathbb{R}^{H \times W})$, while an RGB image has a form of $I \in \mathbb{R}^{H \times W \times 3}$. There are some image formats that use more than 3 channels.

In my thesis I will mostly use this tensor notation when talking about image domain. In some cases though, a different, function based approach will come up, which will be introduced in more detail at that point (see: Boundary loss 5.2.2).

2.2 Computer vision tasks

The most common way to train a deep learning model is to feed it with batches of input data and compare the output with the real target values (ground-truth labels) for that batch. The setup where this kind of labeling of the data is available is called the *supervised learning* setup. In the majority of times I will work with this supervised learning setup throughout the thesis.

Consider an image domain, $\mathbf{I} = \{I_i\}_{i=1}^{N}$, which contains N images. There are many ways to annotate this data to achieve image-label pairs for supervised training. Based on the complexity of the labels we can categorize computer vision tasks. In the following paragraphs, I will introduce one common categorization.

Classification: In this task we want to decide "what is on the picture". Or more formally we have to identify the different objects on the image. Let's say that we have a fixed set of categories, denoted by C. We suppose that for every possible image, there is a ground-truth label-set which is a subset of C. We want to be able to predict this label-set for a previously unseen image. To achieve this, we use the ground-truth label $c \in C$ (or a set of labels $C' \subseteq C$) for each training image of $\mathbf{I}$.

In this case we are only curious about the "content" of the image. For example rotation and mirroring does not have an effect on the ground-truth label. There are other computer vision tasks on the other hand, which have more spatially sensitive annotation.

Object detection: In object detection we want to specify the object(s) position by giving a bounding box as an output. This bounding box gives an area within the subject of interest lies. So in a formal setting, the annotation for the training images are $b \in \mathbb{R}^4$ vectors, which specify the position of the rectangle shaped bounding box. If we also want to classify the localized object as well, we can have annotations in the form of $(c, b) \in C \times \mathbb{R}^4$.

Semantic segmentation: This task is the main topic of this thesis. While in the case of object detection we already cared about the position of the objects in an image, in some application a more detailed spatial description is necessary. For example in self driving cars, the software often times do not only need an approximation of an other car, but the precise position and expanse of it.

That is where semantic segmentation comes in. In this task we basically do classification for every single pixel of the image. That is why sometimes in the literature they refer to it as dense classification. For every image we have a mask for every category with the same resolution as the image. Every pixel in each mask contains the information, whether that pixel belongs to the corresponding class or not. This task often requires different approaches to the previous ones in the deep learning model architectures as well as in evaluation metrics and loss functions.

Instance segmentation: An even more detailed task is the instance segmentation. In this case we do not only have to predict the pixels which belong to the same class, but we have to distinguish between different instances of a class. For example in a photo with multiple people on it, we have to assign a different label for the pixels belonging to different people on the image.

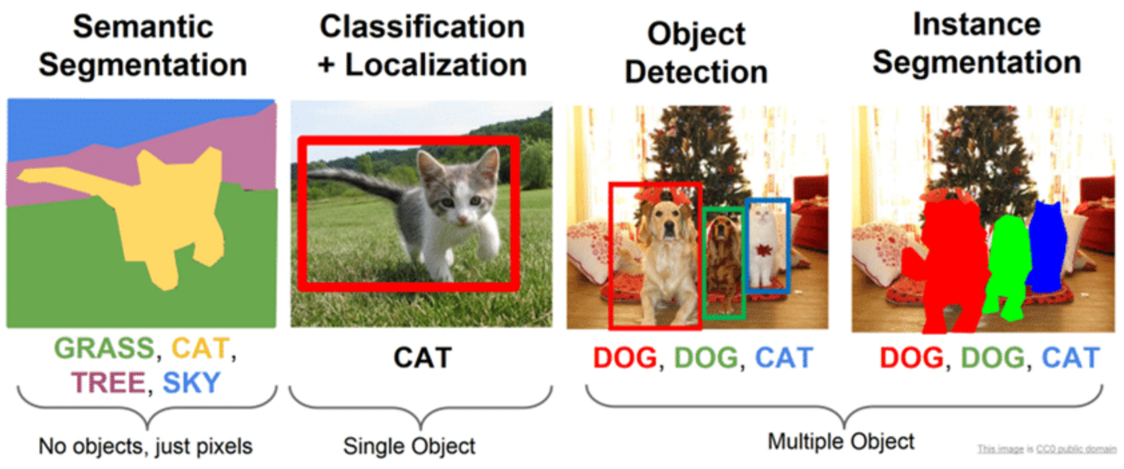

Figure 1: Computer vision tasks [22]

3 Semantic segmentation

3.1 Task description

We have a set of images $\{I_i\}_{i=1}^N \subseteq \mathbb{R}^{H\times W\times Ch}$. The annotations are $\{M_i = (M_{i,1},\ldots,M_{i,C})\}_{i=1}^N$, where $M_i \in \{0,1\}^{H\times W\times C}$. Every $M_{i,k} \in \{0,1\}^{H\times W}$ layer in the tensor indicates the pixels belonging to the k^{th} class. These $M_{i,k}$ tensors are also called (annotation) masks. More formally, for every $1 \le h \le H$ and $1 \le w \le W$, $M_{i,k}[h,w] = 1$ if the pixel $I_i[h,w]$ of the image I_i belons to the k^{th} class, and 0 otherwise.

What we want is a model f, which approximates these masks from the image. The output of the model can either be of $\{0,1\}$ values or real numbers for each pixel. The former method usually involves a thresholding on the raw real valued output. In this case any further calculation and evaluation is done with respect to the threshold parameter applied.

We can define a segmentation model as a parametrized function $f_\theta : \mathbb{R}^{H\times W\times Ch} \to \mathbb{R}^{H\times W\times C}$ (or $f_\theta : \mathbb{R}^{H\times W\times Ch} \to \{0,1\}^{H\times W\times C}$). The parameters θ are coming from the usually huge parameter space Θ of a model family. We want this model f_θ to minimize a certain function between its output and the ground-truth masks of the input images. So our task is to find $\arg\min_{\theta\in\Theta}\big(\operatorname*{aggr}_{i\in\{1,\ldots,N\}} L(f_\theta(I_i), M_i)\big)$. Here L is a function, that expresses a distance between a prediction and the ground truth, which should be minimized. The aggr operation is a way to aggregate these distances across the whole dataset.

There are many ways to define function L and this element often relies heavily on the specific task. Since we are on the field of deep learning, we want to use stochastic gradient descent to find the optimal model. This leads to the distinction between the loss function and the evaluation metrics. Although we want to minimize the average distance between the outputs and the labels defined by L, sometimes we choose a different loss function in the gradient descent algorithm to optimize for. This can be because L might not be directly optimized for as a loss function in gradient descent, because of the differentiability constraint on loss functions. The topic of loss functions and evaluation metrics will be detailed in Section 5.

In the above setup another degree of freedom is f_θ and Θ. These parts are defined through the deep learning model architectures. This topic will be detailed in Section 4.

3.2 Segmentation applications

Probably the most commonly known field related to computer vision and semantic segmentation is self driving cars. The software in these cars must be able to distinguish between the object it sees and based on their classes decide how to act. In order to manoeuvre precisely, it has to be able to segment the different objects around itself.

There are plenty of other fields where a similar technology is greatly applicable. One of these, and the one I will work with the most, is medical imaging. There are many areas in medicine, where doctors and researchers work with images. One can think of chest X-rays, images of the retina or microscopic pictures of cells. All of these examples has related segmentation tasks. Segmenting different organs on chest X-rays, vessels on retina images and cancerous cells on microscopic images are all important and intensively researched areas. As it is demonstrated by these examples, there is a big variation in segmentation tasks even if we only focus on the medical field.

Beside these examples there are other areas which are considered as separate tasks, but are closely related to semantic segmentation. For example in pose estimation we want to determine the position of the people on an image. We do this by assigning a "stick figure" for every person, which estimates the position of their limbs. This is often done by first locating the joints of the human body. By assigning a Gaussian blur around each joint as an annotation for each image, we got ourselves a segmentation problem.

3.3 Challenges in segmentation

In this section I would like to introduce some common challenges and techniques which come up during solving a segmentation task with deep learning. Some of these are originated in classification but has a greater effect in segmentation. Some of them are specific to this task.

Imbalanced data: This problem originates in classification and comes up when one class is heavily overrepresented in the data. It happens even more frequently in segmentation, since even if the distribution of classes is balanced (close to the uniform distribution) across the images, the distribution across pixels can still be very imbalanced. This can occurs when one or more of the classes correspond to objects which are really small in size compared to the scale of the image. For example nuclei in images of tissue or traffic lights on street images. This problem has to be addressed for example with data balancing techniques like over- and undersampling or loss function weighting (see in Section 5).

Noisy labels: Another problem that can also appear in classification, but has a greater impact in segmentation. We talk about noisy labels when the data is labelled incorrectly. It happens in classification tasks as well, but less frequently, because it is usually not so hard of a task to labels whole images. It becomes much harder, when an annotator have to evaluate every single pixel. Segmentation annotation masks are often produced by estimation: someone draws a poligon around the area of interest.

This method, depending on the thoroughness of the annotator, introduces some amount of noise in the label masks. Even if the annotation is done pixel-wise, it is really hard for humans to classify some of these pixels (since the discrete sampling of the continuous real life object through an image). So even in this case, noise can appear in the annotation.

Transfer learning: Because of the time-consuming nature of creating pixel-wise annotation, it is much harder to gather annotated segmentation data compared to classification data. And even if it is available, it can be inaccurate (see previous point). So transfer learning is a very important field for segmentation. It allows us to utilize other datasets and use general features learned on that data to segment the images on which our original problem is defined on.

Data augmentation and generation: Since the scarcity of available training data, augmentation techniques play an important role in segmentation. We can transform our original datapoints to generate new ones. However we have to be careful with this method. While in classification these transformations usually don't change the label, in segmentation they almost always do. Since we have more complex annotations, we must pay more attention on how we transform our data. Beside data augmentation by transformation, the generation of new datapoints is also an important method. Generative models are often used to create new training samples and annotations for segmentation tasks.

Weakly supervised learning: If we are neither able to gather nor to generate labelled training samples with pixel level annotation, then we might have to use weaker labels. This is the field of weakly supervised learning. For example we can utilize training data, where we only have bounding boxes around objects, or only class labels for the whole image. One possible approach in the latter case is the following: Start with classification model trained on the available data. Then we can look at which areas of a given image has the most impact on the final prediction for that image. By finding these pixels we can assume that they correspond to the class, which was predicted by the model. Thus we have a basic segmentation model.

Many models, loss functions or other techniques has been designed or modified only to battle these challenges. During my thesis I will always indicate this and refer to these points when the current subject is related to one or more of these issues.

4 Deep learning models

In this section I am going to provide a broad overview of segmentation models over the past 7-8 years. I am going to start with models using region-of-interest (RoI) suggestions, then I will move on to the newer fully convolutional model solutions. I will put extra emphasis on one special model family namely the U-Net and its different modifications. Finally, I will mention transformers and their applications for computer vision and segmentation.

4.1 Mask R-CNN:

Before convolutional neural networks, models for object detection and classification often used region proposal algorithms. These algorithms select certain regions on the image and suggest them as potential areas which can contain relevant objects. Then we only have to evaluate these regions instead of the whole image.

This technique also appeared in many of the first convolutional neural network (CNN for short) based object detection and segmentation algorithms. The R-CNN model family was probably the most successful one of them for the former task. Taking its most improved version (Faster R-CNN [16]) as a basis, in [17] a segmentation network called Mask R-CNN was proposed, which held the standard for segmentation tasks for a while. I am going to summarize this model in the next paragraphs.

First a standard feature extraction CNN (usually a ResNet version) is applied to gain smaller resolution features from the image. Then this feature map is used for region proposal and the final detection as well.

In the region proposal part a neural network "slides along" the feature maps taking as input a $n \times n$ sized window of the maps. This network predicts for each window a $cls \in \mathbb{R}^{2k}$ and a $reg \in \mathbb{R}^{4k}$ score. Here k is the number of so-called *anchors*. For each $n \times n$ window on the feature maps there is a corresponding area (field of view or FoV for short) on the original image, which affect that region of the feature maps. Anchors are transformations of these FoV areas with one parameter for size scaling and one for height/width ratio. This way the model can propose interesting regions with different size and shape. During training, the output cls and reg scores are compared through a loss function to labels, which are obtained the following way: For every ground-truth object bounding-box, we assign one or more corresponding anchors (ones with high enough intersection with it). Anchors which are assigned to an object have a cls label of 1, while others have 0. Similarly the reg labels are assigned for the anchors based on the object it is assigned to (if there is one). These reg labels will be the ground truth bounding box parameters of the object assigned to an anchor.

In the final model they choose the top N *cls* score and the corresponding *reg* values are proposed as bounding boxes. Based on the proposed bounding boxes, RoIs are cut out from the extracted feature maps as $m \times m$ images. The lattice of pixels on the feature maps and the RoIs are usually not perfectly aligned, so the values of the RoI pixels have to be interpolated (RoIAlign method). Then different *head networks* are applied on the RoIs to predict the class of the object and the segmentation mask. The classification part actually plays an important part in the segmentation head as well. The segmentation head predicts masks for every possible class for the RoI, but in the loss function only the mask of the predicted class is taken into account.

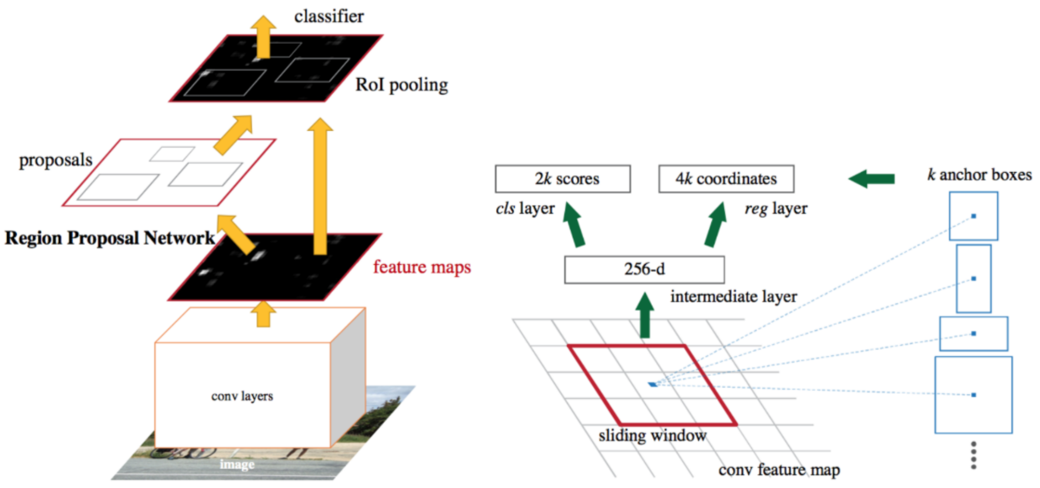

Figure 2: Mask R-CNN region proposal unit [3]

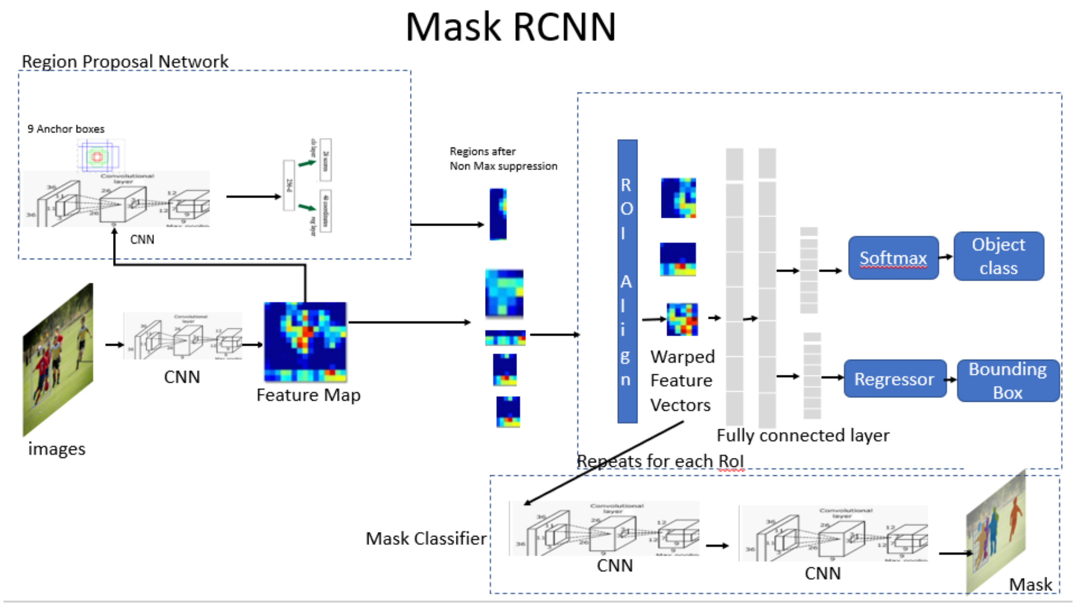

Figure 3: Mask R-CNN model [2]

4.2 Fully Convolutional Networks

A different approach to deep learning semantic segmentation model architectures is fully convolutional networks. In the well known convolutional classification models (like ResNet or VGG) the architecture does not include any region proposal module. The classification is done there by gradually decreasing the spatial size of the image and then converting it to a vector with length equal to the number of classes.

If we want to use the same approach for segmentation we have to obtain an output with the same resolution as the masks, which is the same as the input resolution. The first idea would be to not decrease the resolution of the data during the model (practically speaking, apply convolutions with stride 1, and do not use pooling). The problem with this is that if we want to increase the number of channels (which is a really important feature of deep convolutional networks), then this method requires huge amount of memory for images with higher resolution.

To overcome this problem, the FCN networks were proposed ([25]). These models first apply a series of convolutional and pooling layers to gradually decrease the size of the image and increase the channel size. Then in the second part a reverse process is implemented, where the image size is increased using so called transposed convolutions. This way we get an output with the same resolution as the input and overcome the problem of having to store and compute convolutions for large resolution images with large number of channels.

One extra feature of the model is that in the first part it saves the images before every pooling layer. Then these image tensors are added to the corresponding data with the same resolution in the second part before the transposed convolution. This ensures that the transposed convolution is performed on data coming from two sources. One part went through a "spatial bottleneck" at the end of the first part. So it might have lost some spatial information, but it contains higher level information about the content of the image. The other part, coming through the 'skip-connection', however has gone through less convolutional and pooling layers and thus it has retained more of the original spatial structure of the image.

The description of this model can be the following. Let $\mathcal{C}_{a,b}(x)$ be the function that, given an input $x \in \mathbb{R}^{H\times W\times a}$, outputs a tensor $y \in \mathbb{R}^{H\times W\times b}$ after applying a convolutional kernel and an activation function (or a series of these operations). Let $\mathcal{P}(x)$ denote a pooling function on input x. In the first branch of the FCN model we are going to have a composition of alternating $\mathcal{C}$ and $\mathcal{P}$ functions. Let $d \in \mathbb{N}$ be the depth of the network and let $c = (c_1, \dots, c_d) \in \mathbb{N}^d$ be the number of channels and c_0 be the channel number of the input. Let

$$x^{(0)} = x,\ x^{(1)} = \mathcal{C}_{c_0,c_1}(x) \text{ and } x^{(i)} = f_i(x),\ \text{for } 2 \le i \le d \text{ where}$$

$$f_i = \mathcal{C}_{c_{i-1},c_i} \circ \mathcal{P} \circ \cdots \circ \mathcal{C}_{c_1,c_2} \circ \mathcal{P} \circ \mathcal{C}_{c_0,c_1}.$$

In the following sections I will often use similar chains of function compositions, so the following notation is introduced for a series of functions $g_1, \ldots, g_n$:

$$\bigcirc_{i=1}^{n} g_i = g_n \circ g_{n-1} \circ \cdots \circ g_1.$$

Using this notation we can write the just defined f_i functions as

$$f_i = \bigcirc_{j=2}^{i} (\mathcal{C}_{c_{j-1},c_j} \circ \mathcal{P}) \circ \mathcal{C}_{c_0,c_1}.$$

The first part of the FCN model (which is also called the *downgoing part*) is then defined as

$$f_{down} = \mathcal{C}_{c_d,classes} \circ f_d.$$

Here the extra convolution at the end is applied to get one channel (map) for every possible class.

In the second part of the model (which is also called the *upgoing part*) we are going to apply transposed convolutions to increase the spatial dimensions. This operation is close to being an "inverse" of the convolutional operations in CNN-s. That is why it is often called deconvolution or fractional convolution in the literature. It can be defined as follows.

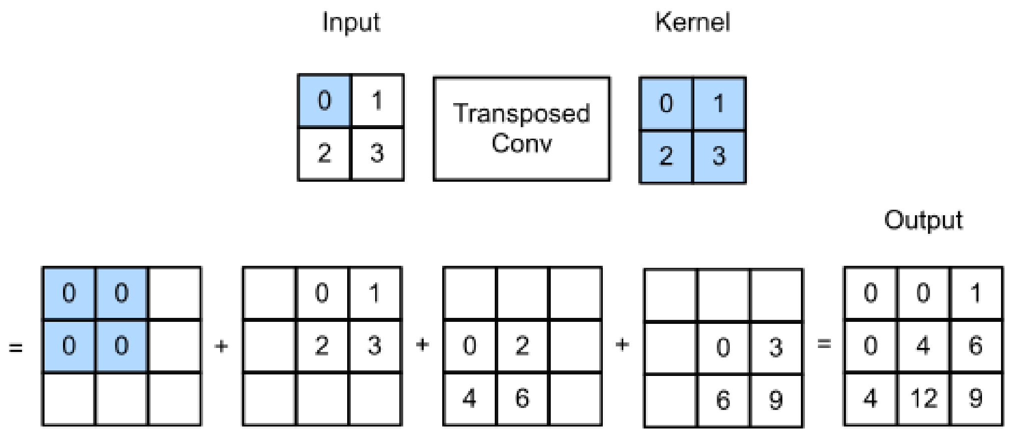

Figure 4: Transposed convolution [1]

Take a one channel $n_1 \times m_1$ image as an input. The transposed convolution operation with stride s and kernel size k transforms this input to a $k + (n_1 - 1)s \times k + (m_1 - 1)s = n_2 \times m_2$ size image by "projecting" the values of the kernel to a $k \times k$ area of the new image. More precisely, a pixel in position (i_1, j_1) in the original image affects the positions $(i_2 + c, j_2 + d)$, $1 \leq c, d \leq k$, in the new image, where $i_2 = (i_1 - 1)s$ and $j_2 = (j_2 - 1)s$. To define the operation itself, do the following: flatten the input image to a vector x with length $n_1 m_1$. Define an $n_1 m_1 \times n_2 m_2$ matrix W as follows. For every (i_1, j_1) pixel in the original image and for every $1 \leq c, d \leq k$

$$W[(i_2 + c - 1)m_2 + (j_2 + d), (i_1 - 1)m_1 + j_1] := K[c, d],$$

where K is the $k \times k$ matrix of the kernel weights. Then the output of the transposed convolution layer is $y = Wx$, reshaped as a $n_2 \times m_2$ image. An illustration of the method can be seen in Figure 4 with an easy example.

Just like regular convolution, transposed convolution can be defined as an operation with multichannel domain and image. Lets denote such an operation by $\mathcal{T}_{a,b}$, similarly to $\mathcal{C}_{a,b}$. Here we are going to use $\mathcal{T}(x) = \mathcal{T}_{classes,classes}(x)$, for brevity. We are also going to use here the $x^{(i)}$ intermediate values of the downgoing part. Using these notations, the upgoing part of the FCN model is defined as

$$f_{up} = \mathcal{T} \circ \oplus_2 \circ \cdots \circ \mathcal{T}(x) \circ \oplus_{d-1} \circ \mathcal{T} = \mathcal{T} \circ \bigcirc_{i=d-1}^{2} (\oplus_i \circ \mathcal{T}),$$

where $\oplus_i(y) = y + x^{(i)}$.

Putting it all together we get the final model:

$$FCN = f_{up} \circ f_{down}.$$

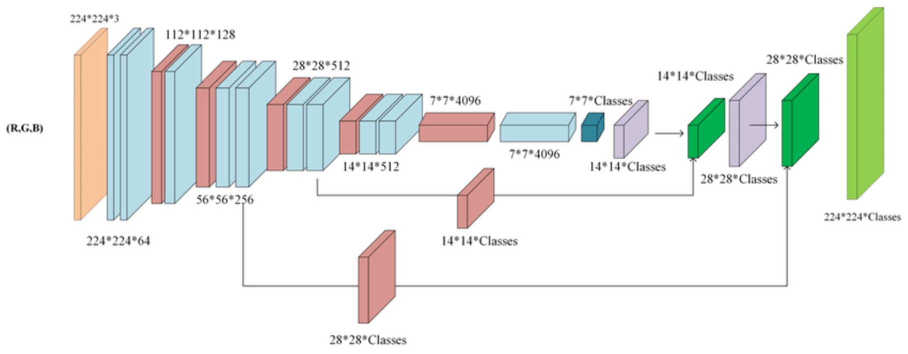

Figure 5: FCN-8 model [27]

4.3 U-Net

The idea of fully convolutional neural networks brought to life many variations of this model architecture. One of the earliest and most successful model was the original U-Net ([28]) for biomedical image segmentation. It became popular mainly because of its great performance of medical image segmentation tasks. The model's structure allows for modifications in the architecture to a great extent. Hence a lots of researchers tried to develop the base U-Net model further by applying newer and newer tricks. In this section I will go through the reasons for the success of U-Nets, the base architecture provided in ([28]) and some popular upgraded versions of the base model.

4.3.1 Base model

The original U-Net model can be viewed as a direct extension of the previously shown FCN model. The main contributions are in the "upgoing" part. This part of the FCN models only used a transposed convolution on the data on each level. In the U-Net on the other hand, we use a so called *upconvolution* on the data of the previous level. Then, after aggregating this with the data coming through the skip connection, another convolutional block is applied. The aggregation is done here by concatenating the two tensor instead of adding them. The channel number here is not reduced after the downgoing part (unlike in FCN), which eliminates an informational bottleneck. We only create the prediction class maps for each class after the upgoing part, in the last convolutional layer of the model.

Formally we can define the U-Net model similarly to the FCN. Let $\mathcal{C}_{a,b}(x)$ be the function, that given an input $x \in \mathbb{R}^{H\times W\times a}$ outputs a tensor $y \in \mathbb{R}^{H\times W\times b}$ after applying a convolutional kernel, an activation function and possibly a batch normalization layer ([18]). Let $\mathcal{B}_{a,b} = \mathcal{C}_{a,b} \circ \mathcal{C}_{b,b}$. Let $\mathcal{P}(x)$ denote a pooling function again. Just like in the FCN model, in the first branch of the U-Net model we are going to have a composition of alternating $\mathcal{B}$ and $\mathcal{P}$ functions. Let $d \in \mathbb{N}$ be the depth of the network and let $c = (c_1, \ldots, c_d) \in \mathbb{N}^d$ be the number of channels and c_0 be the channel number of the input. Let

$$x^{(0)} = x,\ x^{(1)} = \mathcal{B}_{c_0,c_1}(x) \text{ and } x^{(i)} = f_i(x) \text{ for } d \geq i \geq 2, \text{ where}$$

$$f_i = \bigcirc_{j=2}^{i} (\mathcal{B}_{c_{j-1},c_j} \circ \mathcal{P}) \circ \mathcal{B}_{c_0,c_1}.$$

Since we work on with the same channel number after the end of the first part, we simply have the downgoing part of the U-Net as

$$f_{down} = f_d.$$

In the second "upgoing" part we are going to apply alternating upconvolutions and $\mathcal{B}$ convolutional blocks. The upconvolutions, which will increase the spatial dimension, are defined as $\mathcal{U}_{a,b} = \mathcal{C}_{a,b} \circ U$. Here U is the upsampling function, which for an input $x \in \mathbb{R}^{H\times W\times C}$ outputs a tensor with shape $y \in \mathbb{R}^{2H\times 2W\times C}$ by replacing every pixel with a 2 by 2 grid of pixels with the same values as the original pixel.

After the upconvolution, we will concatenate the output with the $x^{(i)}$ values coming through the skip connection. I will call this block of operation as the *aggregation* part and denote it by

$$\mathrm{Agg}_i(y) = (x^{(i)} \mid \mathcal{U}_{c_{i+1},c_i}(y))$$

Here the operation (x, y) denotes the concatenation of tensors $x \in \mathbb{R}^{a\times b\times c_1}$ and $y \in \mathbb{R}^{a\times b\times c_2}$ along the third axis. I.e. $(x \mid y) = z \in \mathbb{R}^{a\times b\times (c_1+c_2)}$ and for $1 \leq i \leq a,\ 1 \leq j \leq b$

$$z[i,j,k] = \begin{cases} x[i,j,k], & \text{when } 1 \leq k \leq c_1 \\ y[i,j,k-c_1], & \text{when } c_1 < k \leq c_1 + c_2 \end{cases}.$$

Using these functions one can define the upgoing part of the U-Net as follows:

$$f_{up} = \bigcirc_{j=d-1}^{1} \left(\mathcal{B}_{2c_j,c_j} \circ \mathrm{Agg}_i\right)$$

After the upgoing part we have to apply one more convolutional layer to obtain the output in the desired shape of $\mathbb{R}^{H\times W\times Classes}$. So to sum it up, the final U-Net model is defined as

$$\mathrm{UNet} = \mathcal{C}_{c_1,classes} \circ f_{up} \circ f_{down}.$$

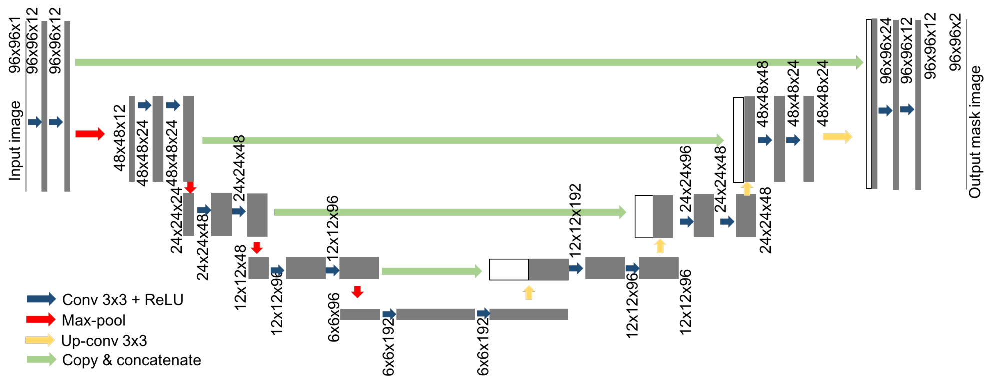

Figure 6: U-Net model [4]

The base U-Net was a great success which motivated a lot of research around this architecture. In the following part I will introduce a successful modification of the original U-Net. This version implemented a more complicated concept in the architecture regarding the aggregation of the lower level and the skip-connection data in the upgoing part.

4.3.2 Attention U-Net

The attention mechanism first came up in the field of natural language processing (NLP). The idea behind this concept is that at a certain point in the model we create weights for the input data and weight it according to these values. This way we can drive the "attention of the model" to the part of the input which is relevant at that state of the process. An example in NLP: say we use a RNN for encoding the input (with refreshed hidden states after every input word). Suppose we use a similar decoder network for creating an output. The decoder takes as input the weighted sum of the hidden states of the encoder. The α_i weights of the i'th word can be determined based on the current hidden state of the decoder and the hidden state of the encoder corresponding to the i'th word.

A similar concept is applicable for the U-Net architecture. Think of the upgoing part of the network as the decoder and the downgoing part as the encoder. Before we concatenate the $x^{(i)}$ values to the data coming through the upconvolution, we can apply a weighting on the pixels of $x^{(i)}$. To determine the weights we shall use $x^{(i)}$ (encoder hidden state) and the values from the lower level of the upgoing part (decoder hidden state).

Lets replace in the upgoing part of the vanilla U-Net the aggregation block with a new attention aggregation block (AttAgg), defined as follows:

$$\text{AttAgg}_i(y) = (\mathcal{U}(y) \mid \mathcal{AB}_i(y)).$$

Here $\mathcal{AB}_i$ is the so-called attention block which creates and applies the weighting of $x^{(i)}$. It has the following parts (I will leave the indexes of $\mathcal{C}$ if that maintains the channel size). First we obtain the so called gating signal with a simple convolution from the lower level data: $\text{GS}_i(y) = \mathcal{C}_{c_{i+1},c_i}(y)$. Then we apply 1×1 convolutions for $x^{(i)}$ and $\text{GS}_i(y)$ and add the two output tensors. (These operations can be done many ways. There is some degree of freedom regarding the spatial size of the two inputs, $x^{(i)}$ and y, that we have to pay attention to.) After this we apply a series of functions to gain a tensor with the same dimensions as $x^{(i)}$. I call this series WMG, short for weight map generation. Then we weight the pixels of $x^{(i)}$ with this by taking the element-wise (Hadamard) product of the two tensors. So formally:

$$\mathcal{AB}_i(y) = x^{(i)} \cdot \text{WMG}(\mathcal{C}(\text{GS}(y)) + \mathcal{C}(x^{(i)}))$$

In the original article they propose WMG $= \mathit{Resample} \circ \mathit{Sigmoid} \circ \mathcal{C} \circ \mathit{Relu}$ and they do not include activation functions after convolutions applied to x and $GS(y)$.

So the upgoing part of the Attention U-Net looks like this:

$$f^a_{up} = \bigcirc_{j=d-1}^{1} (\mathcal{B}_{2c_j,c_j} \circ \text{AttAgg}_j)$$

The downgoing part of the model and the final convolution in the end is the same as in the base U-Net, so all together:

$$\text{AttUNet} = \mathcal{C}_{c_1,classes} \circ f^a_{up} \circ f_{down}.$$

There are many implementation details which can be modified. One arbitrary choice I made is to reduce the channel size in the GS function. It could also be done in the 1×1 convolution following the GS. One more option as I mentioned is the spatial size adjustment of the two input of the attention blocks.

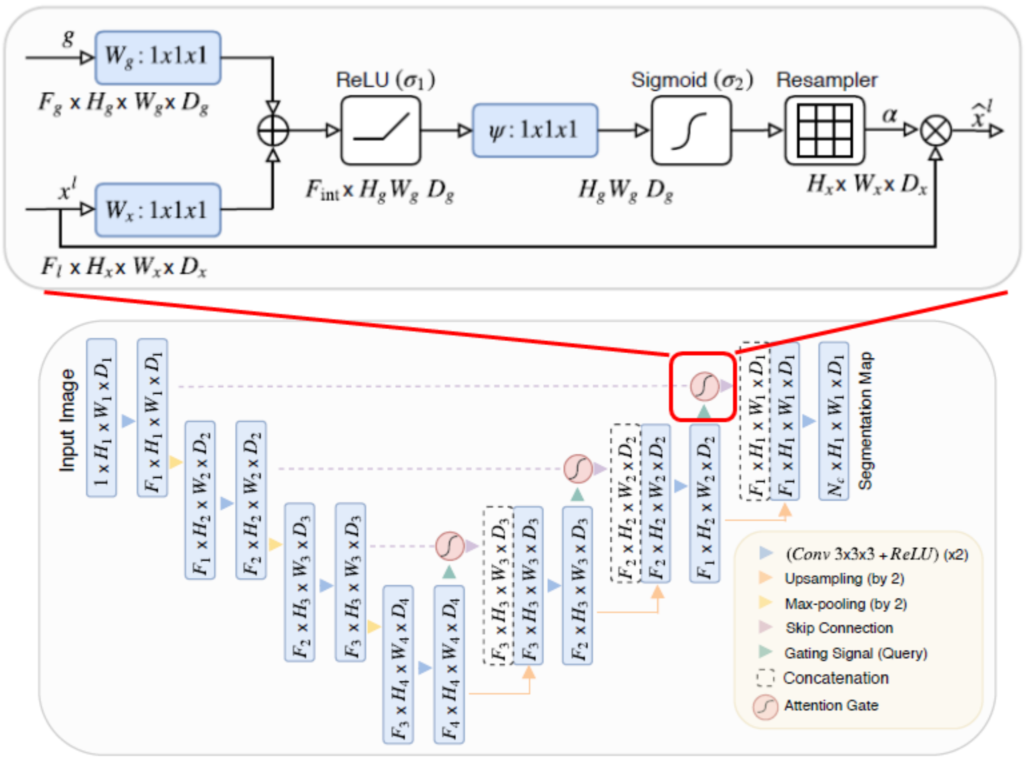

Figure 7: Attention U-Net model and the attention module [26]

4.3.3 Further U-Net variants

A great aspect of the U-Net model is its flexibility. Many parts of the original architecture can be replaced with different or more complex modules leading to a lot of different U-Net variations around the deep learning literature. I just showed one modification in details, where the replaced (or upgraded) part was the aggregation of the lower level and the skip-connection data in the upgoing part. Here I will list and refer to further modification ideas.

Beside the obvious option of choosing the depth and the channel sizes one can modify the convolutional blocks. The original model used a convolutional block with two sequential convolutional layers with ReLU as activation function and a batch normalization layer after each. This unit alone in the model can be defined many different ways. In [14] they used residual connections, while in [5] they defined this block as a recurrent network. Basically if we think of this convolutional block as an independent neural network with a fixed input and output size, we can impute any kind of model as a block that fulfils the input/output requirements.

Just like the convolutional block, the pooling layers in the downgoing part and upconvolutional blocks in the upgoing part can be changed to any neural network with the appropriate input and output shape. There are implementations where they substitute the max pooling with a convolutional layer with stride two, or where they use transposed convolution instead of the upconvolution block.

Obviously one can combine the different modifications, i.e. in [13] they applied residual blocks and the attention mechanism together, which led to a high performing model, the Residual Attention U-Net. A different architectural enhancement is proposed in [36], where they apply a nested U-Net connection with dense skip connections. These, and many more advanced U-Net based architectures are collected in [31].

4.4 Transformers

The first transformer model was published in 2017 and it brought a breakthrough on the field of natural language processing. The model is built up from so called multi-head self attention blocks and so it almost entirely relies on attention mechanism. Thanks to this, the transformer model incorporates context during representation by design. This provided the transformers an advantage to some of the previously used NLP models like recurrent neural networks.

Due to their spectacular results in NLP, there has been many attempts to implement transformer models in computer vision as well. This was not a straightforward task. The multi-head self-attention blocks, in some way, calculate the relationship between the input tokens, and thus the number of operations is cubic in the number of tokens. This is acceptable when the tokens are for example words in a text, but it can lead to problems in image domains if we create tokens from every pixel.

In the breakthrough paper of this area the Vision Transformer (ViT) model was introduced [15]. This model cuts the input image to patches and then applied a linear encoding network to gain input tokens for the transformer block. This model was designed and trained for classification and it achieved the state-of-art result on many benchmark dataset.

It was proved thanks to the ViT model that transformers perform well on image domain. Thus many researchers began to work on transformer models for a wide range of computer vision tasks, including segmentation. Some have tried to directly apply the ViT model as an encoder network in a similar fashion to the classification tasks and either directly predicting segmentation masks from it [32] or applying a more complicated decoder network on top [35]. There were attempts to incorporate transformers to existing segmentation models, like U-Net [12]. And the newer, more complex models [19] can actually perform among the best on popular benchmark datasets, like ADE20K.

5 Metrics and loss functions

For training a deep learning model, one has to choose a loss function to drive the training process and also some evaluation metrics to decide whether the optimization of the hyperparameters are going the right way. These two components are closely related in the framework. Sometimes we can use the same thing for them, but it is generally not a good idea. It can happen that we want to measure the performance with a metric function that is hard to optimize for as a loss function (possible reasons are not smooth enough derivative, or differentiability at all). If we try the other way and only evaluate a set of hyperparameters based on the loss function value, then we might add an unintended bias to the final model, that will ruin the performance when the model is launched in its intended final environment.

For these reasons, there is a significant emphasis on researching proper loss and metric functions in semantic segmentation as well. In this chapter I would like to provide a non-exhaustive review of some of these functions, which were found useful in certain segmentation usecases.

5.1 Metrics

A really important part of a deep learning framework is how we evaluate the current model and set of hyperparameters. We rely on these metrics a lot, since they determine the direction of the model and training development. There are many ways to evaluate a deep learning segmentation model. Many of these evaluation metrics are inherited from the classification tasks. A popular set of such metrics in the case of binary classification use the elements of the confusion matrix. The only difference is that in the case of segmentation we do not only sum (average) over the training samples, but also over the pixels in each sample.

In this section I will mainly focus on the two class (binary) case, where we want to classify pixels into two possible classes. Most metrics can be calculated for the multiclass case by calculating one vs rest binary scores for each class and then taking the average.

We define the confusion matrix for the output of the segmentation model with two classes as like we applied a binary classification for every pixel of the input image. This way we get for example the TP value by counting every pixel that was classified correctly (doing similarly for FP, FN, TN).

To use these in the traditional way, we can't have continuous outputs. We need to threshold the raw output of the models to determine whether a pixel belongs to a certain class or not. In this part lets look at the output of the network as a vector instead of a matrix. So lets say, that the network output is in the from of $\hat{y} \in \mathbb{R}^p$ and $y \in \{0,1\}^p$ is the ground-truth vector. Let $T_\delta : \mathbb{R}^n \to \{0,1\}^n$ be the thresholding function such that if $T_\delta(x)_i = 1$ if $x_i > \delta$ and 0 otherwise. Then we define the following metrics:

$$TP = y^T \cdot T_\delta(\hat{y}),\ FP = (\mathbb{1}-y)^T \cdot T_\delta(\hat{y}),\ FN = y^T \cdot (\mathbb{1}-T_\delta(\hat{y})),\ TN = (\mathbb{1}-y)^T \cdot (\mathbb{1}-T_\delta(\hat{y})).$$

From these, one can calculate a whole family of confusion matrix based metrics. Two of the most popular ones are the Sorensen-Dice coefficient (DSC) and the Jaccard index:

$$\text{DSC} = \frac{2TP}{2TP + FP + FN}$$

$$\text{Jaccard index} = \frac{TP}{TP + FP + FN}$$

Most of the metrics defined this way has one disadvantage: they rely on the threshold value, δ. The following two metrics aim to avoid this. These are the ROC an PR curves and their AUC metrics. The former curve pictures the $TPR = \frac{TP}{TP+FN}$ and $FPR = \frac{FP}{FP+TN}$ values against each other with all the possible threshold choices. The PR curve, however, pictures the $\mathit{precision} = \frac{TP}{TP+FP}$ and $\mathit{recall} = \frac{TP}{TP+FN}$ values the same way. Using a ROC is a good choice with balanced classes in the training data. However, in case of imbalanced data it can overestimate the performance of the model. It is because FPR can be really small if TN is significantly larger than FP. This leads to great ROC AUC scores for models where the $TN \gg FP \gg TP$ which is a really undesired behaviour. PR curves avoid this problem by not using the TN values at all [29]. The area under the PR curve is also called the average precision score (or AP for short).

If we look at segmentation predictions and ground truth masks as subsets of the plane (see this concept in more detail in Section 5.2.2), then we can define some entirely different metrics for the task. One way to measure how far two subsets of the plane ($X, Y \subseteq \mathbb{R}^2$) are from each other is the Hausdorff distance:

$$d_H(X,Y) = \max\left\{\sup_{x\in X} d(x,Y), \sup_{y\in Y} d(y,X)\right\} = \max\left\{\sup_{x\in X}\inf_{y\in Y} d(x,y), \sup_{y\in Y}\inf_{x\in X} d(y,x)\right\}$$

In our case X and Y are going to be the sets indicated by the model prediction and the ground truth masks, so they are discrete, finite subsets of $\mathbb{R}^2$. Thus the supremums and infimums above are actually maximums and minimums.

Using this method of evaluation we always take into account the largest mistakes. This way we can get bad Hausdorff distance scores because of a few far off predictions. Sometimes it is a desired feature of the metric. For example if we really want to avoid prediction positive pixels far from the ground truth area. On the other hand, this means that a few badly predicted pixels can ruin the score of an otherwise great model. To avoid this, the average Hausdorff distance (AHD) can be applied:

$$d_H(X,Y) = \frac{1}{2}\left(\frac{1}{|X|}\sum_{x\in X} d(x,Y) + \frac{1}{|Y|}\sum_{y\in Y} d(y,X)\right).$$

Some argue, that averaging over the prediction set in the first term is a bad idea and we should instead divide by the size of the ground truth set in both terms (balanced average Hausdorff distance [7]). We can also define $d(x,Y)$ to be the 95th percentile of $d(x,y)$ distances in decreasing order, insead of $\min_{y\in Y} d(x,y)$. I will call this version the modified average Haussdorf distance (MAHD).

5.2 Loss functions

Just like in any other deep learning task, choosing the appropriate loss function for our model is really important. In semantic segmentation there has been several approaches to create new and better loss functions. I will introduce some of them in the next section.

The first approach is to think of semantic segmentation as a dense classification task. In this scenario we can simply apply losses which are common in classification tasks, like cross-entropy. For one prediction-label pair ($y, \hat{y} \in \mathbb{R}^C$) the cross-entropy loss function looks like this:

$$L_{CE}^{class}(\hat{y}, y) = \sum_{i=1}^{C} y_i \log(\hat{y}_i),$$

where y_i is the indicator of class i in the image and $\hat{y}_i$ is the prediction for class i.

If we want to use this for segmentation, we have to integrate over the pixels of the image ($y, \hat{y} \in \mathbb{R}^{h\times w\times C}$):

$$L_{CE} = L_{CE}^{segm} = \frac{1}{h\cdot w}\sum_{h,w}\sum_{i=1}^{C} y_i[h,w]\cdot\log(\hat{y}_i[h,w])$$

In some cases we want to apply weights for the different classes (for example in case of an imbalanced dataset):

$$L_{WCE} = \frac{1}{h\cdot w}\sum_{h,w}\sum_{i=1}^{C} w_i\cdot y_i[h,w]\cdot\log(\hat{y}_i[h,w])$$

Another interesting modification of this loss is the so-called Focal loss [23]. This version puts more emphasis (with parameter $\gamma \in \mathbb{R}$) for datapoints which are hard for the current model to classify, i.e. datapoints with high corresponding loss terms:

$$L_{focal} = \frac{1}{h\cdot w}\sum_{h,w}\sum_{i=1}^{C}(1-\hat{y}_i[h,w])^{\gamma}\cdot y_i[h,w]\cdot\log(\hat{y}_i[h,w]).$$

Now lets restrict ourselves to two classes ($|C| = 2$). Then one can also take metrics derived from the confusion matrix (like the DSC or Jaccard index) score and use these as a loss function. However, if we want to directly optimize for these metrics, we have to make some modifications to the definition, since in the original form the total number of correctly (falsely) classified pixels are counted after applying a threshold function, which is not differentiable. Because of this, from now on I will use the following notions for the upcoming losses.

Definition. *If $y \in \{0,1\}^p$ is the ground-truth vector for an input image and $\hat{y} \in [0,1]^p$ is the model output, then*

$$TP = y^T\hat{y},\ FP = (\mathbb{1} - y)^T\hat{y},\ FN = y^T(\mathbb{1} - \hat{y}),\ TN = (\mathbb{1} - y)^T(\mathbb{1} - \hat{y}).$$

These function are the generalization of the previously defined conf matrix values, since they have the same values for $\hat{y} \in \{0,1\}^p$. If we have these generalized values we can safely define the following loss functions.

$$\text{DiceLoss}(y,\hat{y}) = 1 - \frac{2 \cdot TP + \varepsilon}{2 \cdot TP + FN + FP + \varepsilon}$$

$$\text{JaccardLoss}(y,\hat{y}) = 1 - \frac{TP + \varepsilon}{TP + FN + FP + \varepsilon}$$

$$\text{TverskyLoss}(y,\hat{y}) = 1 - \frac{TP + \varepsilon}{TP + \alpha FN + \beta FP + \varepsilon}$$

Here ε is added to avoid edge cases like division with 0.

The Tversky loss is a common generalization of the previous two. For $\alpha = \beta = 1$ we get exactly the Jaccard loss and for $\alpha = \beta = \frac{1}{2}$ we get $\frac{1}{2} \cdot \text{DiceLoss}$. The α and β parameters in this loss gives us the option to weight the FN and the FP rate accordingly. If we assign more weight for false negatives, then the result will tend to overestimate the positive region, since it will try to avoid predicting positive pixels as negatives. This can be advantageous for example in many medical areas, where having a false negative diagnosis has way worse consequences, than having a false positive.

5.2.1 Lovász loss

We have seen in the previous section how to define loss functions based on some classification metrics. There has been some deeper research though, on how to extend these discrete domain metrics for model outputs with continuous values, to make for a better loss function. In the following part I will introduce one such result regarding the Jaccard index.

For this I first have to introduce the *Lovász-extension* of real valued set-functions. Here I will use notations greatly inspired from the notations in [8]. For any set function with the domain of 2^V for some fixed set V, $|V| = p$, one can define the same function on the characteristic vectors of the subsets of V. So if we have a set function F, I will use both of the following notions: for $S \subseteq V,\ s \in \{0,1\}^p,\ F(S) = F(s)$ if s is the characteristic vector of S.

If we have a set-function $F : \{0,1\}^p \to \mathbb{R}$ for a fixed pointset V ($|V| = p$), then its Lovász extension is a $f : \mathbb{R}^p \to \mathbb{R}$ piecewise linear function such that $f(s) = F(s)$ for every $s \in \{0,1\}^p$. The following definition shows, how the Lovász-extension is constructed.

Definition. *Let $F : \{0,1\}^p \to \mathbb{R}$ be a set-function, s.t. $F(\emptyset) = 0$. The Lovász-extension of F is the function $f : \mathbb{R}^p \to \mathbb{R}$:*

$$f(x) = \sum_{i=1}^{p} x_{\pi_i} g_i(x),$$

where $g_i(x) = F(\sum_{k=1}^{i} e_{\pi_k}) - F(\sum_{k=1}^{i-1} e_{\pi_k})$. Here e_i is the i^{th} unit vector, and $\pi = (\pi_1, \ldots, \pi_p)$ is the permutation such that $x_{\pi_1} \geq \cdots \geq x_{\pi_p}$.

There are many equivalent ways to formulate this definition (see for example Definition 3.1. in [8]).

Now turn onto the connection to loss functions. The result I am about to introduce is the *Lovász loss* [9]. Lets denote the ground truth values by $y \in \{0,1\}^p$ for a given input X, and the set of positive pixels by P. We can define the Jaccard index as a function of the set of misclassified pixels by a model. For a given M set of misclassifications $F_J(M) = \dfrac{|M|}{|M \cup P|} = 1 -$ Jaccard index. We can now use the Lovász-extension of this set function as a loss function of a segmentation model. Lets denote the raw unthresholded output of the model by $\hat{y} \in [0,1]^p$ and let $m_i = |y_i - \hat{y}_i|$.

$$\text{LovaszLoss}(y, \hat{y}) = f_J(\boldsymbol{m}(y, \hat{y})),$$

where f_J is the Lovász-extension of F_J.

What is the advantage of this construction over the previously shown one? If it feasible in computational time? These questions are answered in the original article of the Lovász-softmax loss [9]. For the second question they explained that the extension of F_J can be computed in $p \log p$. For the first question to answer, we need a longer argument.

First of all I will define submodularity in terms of set-functions:

Definition (Submodularity)**.** *A set function $F : 2^V \to \mathbb{R}$, $|V| = p$, is said to be submodular if for every $A, B \subseteq V$*

$$F(A) + F(B) \geq F(A \cap B) + F(A \cup B). \tag{1}$$

Submodularity can be formulated in many equivalent ways (see Section 2.1. in [8]), for example we can use the first order differences.

Definition (Submodularity 2)**.** *A set function F on V is submodular if for every $A \subseteq B \subseteq V$ and $i \notin B$:*

$$F(A \cup \{i\}) - F(A) \geq F(B \cup \{i\}) - F(B). \tag{2}$$

The Lovász-extension plays a significant role in submodular function analysis. This is in a big part due to the fact, that submodularity in set-functions is a similar phenomena in some sense as convexity in real valued functions and the connection between the two can be described with the Lovász extension.

Theorem 1 (Proposition 3.6 of [8])**.** *A set-function F is submodular if and only if its Lovász extension f is convex.*

Proof. (Sketch) First assume that f is convex and take two arbitrary sets $A, B \subseteq V$. Take $f(\mathbb{1}_{A\cup B} + \mathbb{1}_{A\cap B})$ (which is equal to $f(\mathbb{1}_A + \mathbb{1}_B)$). Using an equivalent definition of the Lovász-extension (Prop. 3.1. in [8]) we can reformulate it as

$$f(\mathbb{1}_{A\cup B} + \mathbb{1}_{A\cap B}) = \int_0^2 F(\{(\mathbb{1}_{A\cup B} + \mathbb{1}_{A\cap B}) \geq z\})dz = F(A \cup B) + F(A \cap B).$$

If we use the convexity of f and that $F(A) = f(\mathbb{1}_A)$ we get

$$F(A\cup B)+F(A\cap B) = f(\mathbb{1}_{A\cup B}+\mathbb{1}_{A\cap B}) = f(\mathbb{1}_A+\mathbb{1}_B) \leq f(\mathbb{1}_A)+f(\mathbb{1}_B) = F(A)+F(B),$$

which is exactly what we needed.

For the other direction we can use a proposition (Prop. 3.2 in [8]), which states that if F is submodular, then f can be written as point-wise maximum of linear functions, which means that f itself is convex.

$\square$

This has important consequences, since convexity plays a really significant role in optimization. If we have a submodular metric function for i.e. a dense classification task, then using its Lovász extension (or the negative of it) as a loss function leads to a convex optimization task (with respect to the model output). This make for a faster and easier learning of the model.

Inspired by this logic, the submodularity of the Jaccard index would validate the use of the above defined LovaszLoss over the JaccardLoss.

Theorem 2. *$F_J(M)$ is a submodular set function.*

Proof. We have to show, that (2) holds. Suppose we have two sets of mispredictions $A \subseteq B$ and a pixel $i \notin B$. Lets say that TP_M, FP_M, FN_M denote the number of true positive, false positive and false negative pixels respectively in case of a misprediction set M. This way we have that

$$F_J(M) = \frac{FP_M + FN_M}{TP_M + FP_M + FN_M}.$$

Assume now that the extra pixel i is false negative. Then we have

$$F_J(B \cup \{i\}) = \frac{FN_B + 1 + FP_B}{TP_B - 1 + FN_B + 1 + FP_B} = F_J(B) + \frac{1}{TP_B + FP_B + FN_B}. \tag{3}$$

We can also observe the following:

$$A \subseteq B \Rightarrow FP_A \leq FP_B \textit{ and } FN_A \leq FN_B \tag{4}$$

Using (3) and (4) we obtain that

$$\begin{aligned} F_J(B \cup \{i\}) - F_J(B) = \frac{1}{TP_B + FP_B + FN_B} \leq \\ \leq \frac{1}{TP_A + FP_A + FN_A} = F_J(A \cup \{i\}) - F_J(A). \end{aligned} \tag{5}$$

Assume now that the extra pixel i is false positive. Following a similar argument we obtain a bit more complicated case since the new false positive pixel comes from the previously uncounted true negative set.

$$\begin{aligned} F_J(B \cup \{i\}) - F_J(B) &= \frac{FN_B + FP_B + 1}{TP_B + FP_B + 1 + FN_B} - \frac{FN_B + FP_B}{TP_B + FP_B + FN_B} = \\ &= \frac{TP_B + FN_B - FN_B}{(TP_B + FN_B + FP_B + 1)(TP_B + FN_B + FP_B)} \leq \\ &\leq \frac{TP_B + FN_B - FN_A}{(TP_B + FN_B + FP_A + 1)(TP_B + FN_B + FP_A)} = \\ &= F_J(A \cup \{i\}) - F_J(A). \end{aligned} \tag{6}$$

The inequality comes from (4) again. The last equality holds because the union of true positives and false positives give the whole set of positive pixels, which does not depend on the prediction set, so $TP_B + FN_B = TP_A + FN_A$.

□

This theorem supports the idea of using Lovász loss instead of the original Jaccard loss when we are trying to optimize directly for the Jaccard index.

5.2.2 Boundary loss

To define different losses to the ones that I already detailed, I am going to use here a different approach to the segmentation task. I already mentioned this in previous sections, but now I will detail it. Lets think about the GT masks as a region (set) or a curve in $\mathbb{R}^2$. We are trying to approximate this region with our model. If we assume that the output of the model is also a subset of $\mathbb{R}^2$, than we can use the different metrics to measure their similarity. For example using the Hausdorff distance, like in [20], we can define and train models in this manner.

One advantage of these techniques is that they are usually less sensitive for extreme inbalance in the data then the traditional regional losses. In these methods integrals are usually taken over the boundary of the target area rather than the whole image, which leads to this property.

The result I want to highlight here is the so called Boundary loss. Using the notion of [21], define the training image as a function and denote it by $I : \Omega \subseteq \mathbb{R}^2 \to \mathbb{R}$. We will give the ground truth masks and the predictions as subsets of the domain of I, Ω. Lets define the ground-truth annotation as $G \subseteq \Omega$ and denote the indicator of G by $g : \Omega \to \{0, 1\}$, i.e. for $x \in \Omega$ $g(x) = 1$ if $x \in G$ and $g(x) = 0$ otherwise. Let $s : \Omega \to [0, 1]$ denote the raw output of the model and s_δ be the thresholded 0-1 output for some δ. Let $S_\delta \subseteq \Omega$ be the set indicated by s_δ. We also assume that S and G are closed and connected for the followings to work.

By defining a metric between the boundaries $\partial G, \partial S_\delta$ we can measure the similarity of the two regions. An evident way to do this is by integrating along the distances of the border points. Let

$$\mathrm{Dist}_1(\partial G, \partial S_\delta) = \int_{\partial G} ||N(x)||^2 dx.$$

Here $N(x) = y_{S_\delta}(x) - x$, where $y_{S_\delta}(x)$ is the nearest point of ∂S to x on the line of the normal vector of ∂G in x (we suppose that it exists for every x). In [21] and [10] it is shown that this distance is closely related to the following other distance notion of two set boundaries.

$$\mathrm{Dist}_2(\partial G, \partial S_\delta) = \int_{G \Delta S} d(x, \partial G) dx.$$

Here $d(x, G)$ is the euclidean distance between x and G.

Boundary loss can be derived from the second one, so from now on $\mathrm{Dist}(\partial G, \partial S_\delta) = \mathrm{Dist}_2(\partial G, \partial S_\delta)$.

Lets denote the signed distance function from ∂G by D_G, i.e. $D_G(x) = -d(x, \partial G)$ if $x \in G$ and $D_G(x) = d(x, \partial G)$ otherwise. Using this we can rewrite

$$\begin{aligned} \mathrm{Dist}(\partial G, \partial S_\delta) = \int_{G \Delta S} d(x, \partial G) dx &= \int_S D_G(x) dx - \int_G D_G(x) dx = \\ &= \int_\Omega D_G(x) s_\delta(x) dx - \int_\Omega D_G(x) g(x) dx. \end{aligned} \tag{7}$$

The last term does not rely on the output of the model. If we replace $s_\delta(x)$ with $s(x)$, then we get a differentiable function of the output which (aside from a

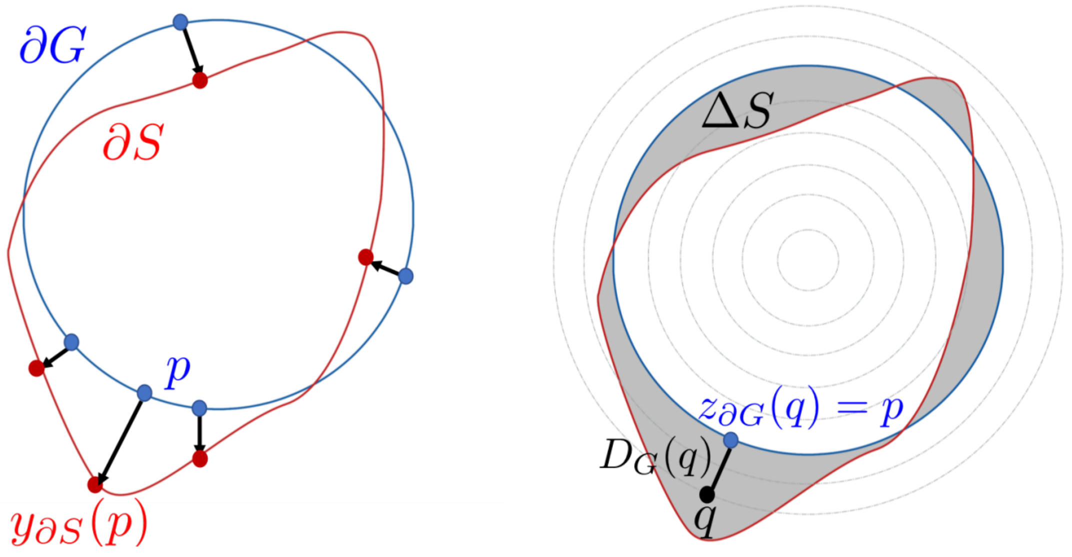

Figure 8: Illustration of Dist_1 and Dist_2 [21]

constant) approximates the distance between the predicted area boundary and the ground truth boundary. This term is what they call Boundary loss:

$$\text{BoundaryLoss}(y, \hat{y}) = \int_\Omega D_G(x)s(x)dx,$$

where G is the set indicated by y and s is calculated directly from $\hat{y}$.

In practice the $D_G(x)$ distance maps are calculated in advance, since it does not depend on the model. During training we use it as an additional channel to the ground-truth annotation mask so it does not increase the training time significantly.

Following the recommendations of [21] I tried the Boundary loss as part of a combined loss function. There are many ways to do this. One is when we add a second loss with some parameter $\alpha \in \mathbb{R}$:

$$L(y, \hat{y}) = L_1(y, \hat{y}) + \alpha L_2(y, \hat{y}).$$

Another fairly similar way is to have the convex combination of the two losses:

$$L(y, \hat{y}) = (1 - \alpha)L_1(y, \hat{y}) + \alpha L_2(y, \hat{y}).$$

We can choose α to be a constant or we can modify it as the training progresses. In the latter case α is the function of the current epoch number k. Popular functions for the combination types are $\alpha(k) = k \cdot \alpha_0$ for the first combination type and $\alpha(k) = 1 - \frac{1}{e^{\beta \cdot k}}$ for the second combination type ($\beta \in [0, 1]$).

Some modifications can also be done to the distance maps. By clamping, rescaling or powering the values, we have the ability to push the learning procedure to a desired direction. For example we can achieve a higher or lower emphasis for pixels around the border. Or we can say that pixels further from the border than a certain value should not be penalized more.

6 Application

To demonstrate the previously detailed models, loss functions and metrics I am going to present some result in this section. I have conducted experiments on two different medical image dataset. One is publicly available , which contains lung X-ray images. The other is not a public dataset and it contains histopathological images of lung tissues. First I will introduce the two dataset in more details and then I will present the results of my experiments.

6.1 Datasets

6.1.1 COVID-QU-Ex

The COVID-QU-Ex dataset ([6]) consists of 33 920 chest X-ray (CXR) images and their corresponding annotations. We have images with resolution of 256x256 pixels. 11 956 X-rays are from COVID-19 infected patients, 11 263 are from non-COVID (viral or bacterial pneumonia) infected patients and 10 701 are from healthy patients.

We have two type of annotation masks for the dataset. For every image we have a mask that indicates the lung on the X-ray image (lung segmentation mask). This gives us 33 920 image-mask pair for lung segmentation.

For a subset of the COVID-19 infected patients we also have annotation masks for the infected area of the lung (infection segmentation mask). This gives us 2 913 image-mask pair for infection segmentation.

A sample of the dataset is shown in Figure 9 with the original X-Ray, the corresponding infection mask, and the lung segmentation mask.

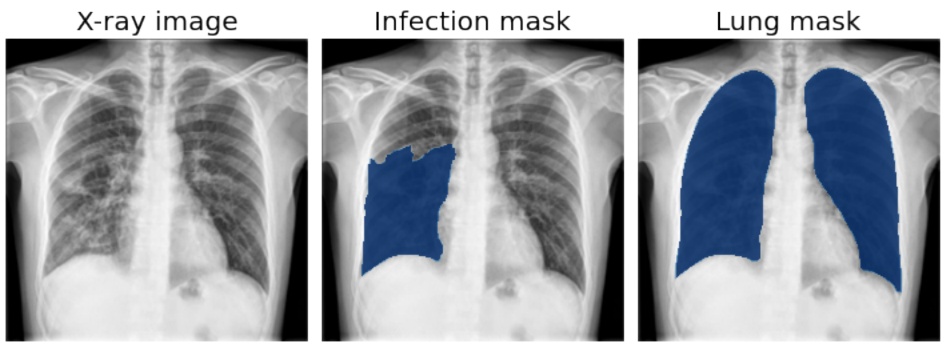

Figure 9: The X-ray image, the COVID-19 infection mask and the lung mask.

6.1.2 Korányi histopathology dataset

The second dataset consists of histopathological images. These are really large resolution images (WSI) of lung tissues. We have a total of 7 of 41 984 by 89 344 slides. On three of the slides we have cancerous tissue parts. Two of them contains tissue with Squamous-cell carcinoma (SCCs), and the third has lung adenocarcinoma (ADC). As annotation we have masks, which denotes for each pixel of the slides whether they belong to a cancerous region of the tissue or not, and if it does, which type of cancer it is (see an example in Figure 10). During my experiments I mainly worked on the two ADC slides. Thus I had two 41 984 by 89 344 image with the annotation masks.

During the preprocessing part these images had to be reduced. The two WSI images were cut up to 512x512 pixel images. Many of these smaller images (patches) had to be thrown away because it didn't contain any tissue. From the rest I only used those, which pictured tissues with cancerous regions. This approach was chosen because the dataset is really inbalanced (Figure 11). The after throwing away the patches with no tissue, only 0.6% of the pixels were positive (belonged to cancerous regions). This was increased to 15.9% by using only the positive patches (which contain any positive pixels).

The models which I used took input images with size 256x256 pixels. These input images were extracted from the 512x512 patches by applying a so-called random pick operation (randomly choosing a 256x256 region of the patch, which can even be rotated with a random angle). This procedure also acted as an augmentation tool on the data.

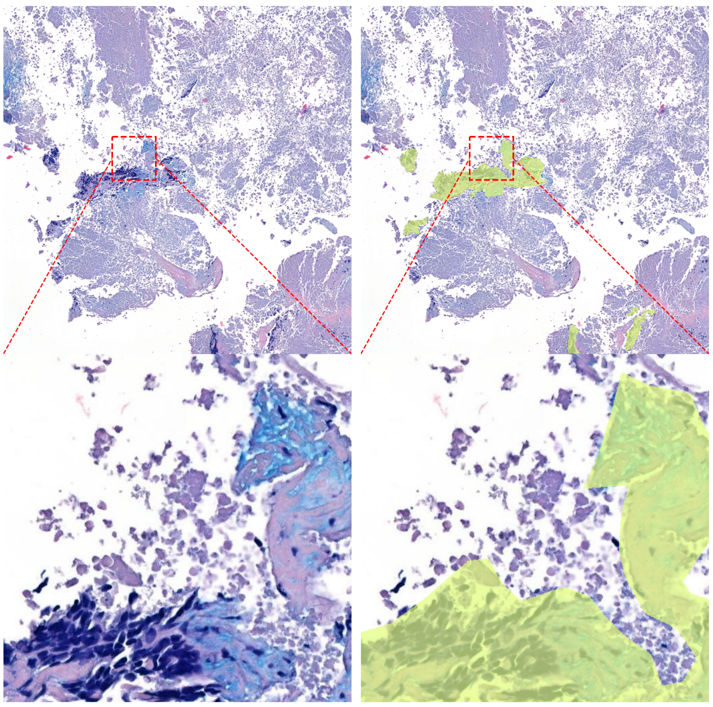

Figure 10: Top: WSI (4096x4096), bottom: cropped patches (512x512), left: original image, right: annotation

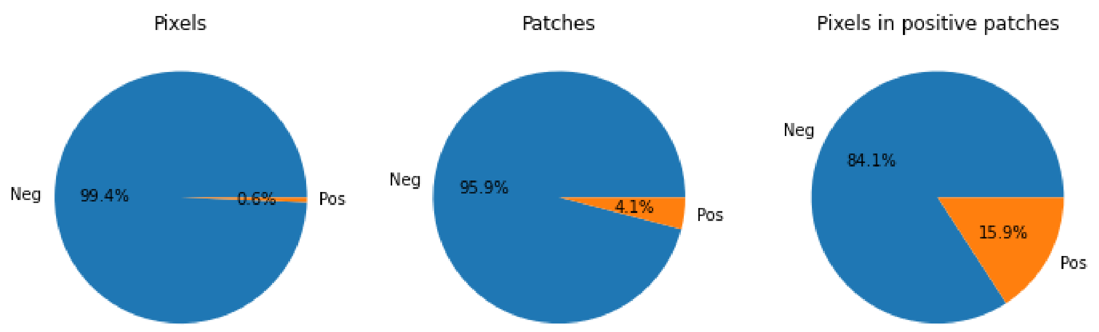

Figure 11: Ratio of positive pixel and patches in the histopathology dataset

6.2 Experiments and results

I am going to present the results of experiments about the performance of different U-Net models, the advantages of Boundary loss and the benefits of Lovász loss over Jaccard loss. You can find the detailed setups and hyperparameters for each in the Appendix. There I will also show images of the predictions of the models.

I will measure the performance with different metrics, all of which have been mentioned before in Section 5.1 (AUC will always indicate ROC AUC). The arrows indicate whether the higher or the lower values are better.

6.2.1 Technical setup

The experiments ran on a computer with Ubuntu 20.04.4 LTS operating system. Every experiment used an NVIDIA GeForce RTX 2080 Ti GPU unit with 12 GB memory. I used Python (Python 3.9) with PyTorch (PyTorch 1.8) for model building and training.

The experiment were run in a general purpose deep learning pipeline. There was no separate test set used, the presented values are the validation results. I only wanted to demonstrate the previously introduced techniques and the validation results can serve this purpose.

6.2.2 Results

Experiment 1

In this experiment I compare the performance of three different U-Net variants. These are the vanilla U-Net, the Attention U-Net and the Residual Attention U-Net. The experiments were conducted on the COVID-QU-Ex infection segmentation task. The results are presented in Table 1.

From these results we can see very little differences between the models, often not in the favor of the more complex models. Almost all differences are within standard deviation range (calculated from 5 different runs with different random

	loss	val loss	BA	AUC	AP	DSC	Jaccard	MAHD
UNet	0.0199 (0.0021)	0.0395 (0.0017)	0.8928 (0.0079)	0.9461 (0.0134)	0.8078 (0.0146)	0.8156 (0.0041)	0.6942 (0.0064)	0.2414 (0.0059)
AttUNet	0.0192 (0.0016)	0.04 (0.0017)	0.8973 (0.0086)	0.9519 (0.0073)	0.8136 (0.0167)	0.8131 (0.0043)	0.6907 (0.0061)	0.2399 (0.0096)
ResAttUNet	0.0143 (0.0018)	0.0352 (0.0039)	0.8977 (0.0157)	0.9374 (0.0141)	0.8124 (0.0203)	0.8235 (0.0116)	0.7042 (0.0173)	0.2271 (0.0241)

Table 1: U-Net model comparison

initialization). From these values we can deduce that on this dataset and task, more advanced models are not necessarily the key of improvement. Probably more training data and maybe more hyperparameter tuning is the more rewarding approach.

Experiment 2

In the second experiment I would like to compare the results of the Jaccard loss and the Lovász loss on the COVID-QU-Ex dataset. In Table 2 you can see the results of infection segmentation experiments with the two losses.

	AUC ↑	DSC ↑	Jaccard index ↑	average precision (AP) ↑
Jaccard	0.9085	0.8106	0.6883	0.7905
Lovasz	0.9211	0.8297	0.7132	0.7999

Table 2: Lovasz vs Jaccard

We can see a slight but clear advantage in favor of the Lovász loss, which is what we expected from Section 5.2.1. This is promising, but more experimenting are necessary to state that the Lovasz loss clearly outperforms the Jaccard loss in practice.

Experiment 3

In my final experiment I will explore the performance of the Boundary loss combined with some other losses (BCE and Tversky). In Table 3 you can see the results of the pure boundary loss, the pure tversky loss and the combination of the two (with linear and with constant combination) compared to each other. The same can be seen for BCE and Boundary in Table 4 (only with constant combination).

	MAHD ↓	DSC ↑	Jaccard index ↑	average precision (AP) ↑
Boundary	0.3460	0.6113	0.4564	0.8097
Tversky	NaN	0.6187	0.4744	0.6102
Combined (constant)	0.2712	0.7495	0.6150	0.7960
Combined (linear)	0.2427	0.7047	0.5718	0.7951

Table 3: Tversky and Boundary combination

	MAHD ↓	**DSC ↑**	**Jaccard index ↑**	**average precision (AP) ↑**
Boundary	0.3460	0.6113	0.4564	0.8097
BCE	0.2281	0.7505	0.6235	0.8669
Combined (constant)	0.1962	0.7842	0.6645	0.8839

Table 4: BCE and Boundary combination

We can see that the combination of Tversky and Boundary clearly outperforms both of these losses alone, which shows the power of loss combination in case of Boundary loss. We see the same with a bit smaller differences in the case of BCE and Boundary loss. These results support the usability of the Boundary loss in case of noisy annotations like the ones in the Korányi dataset.

Future research

I am planning to continue my investigation on the presented datasets and about the topics I have introduced in my thesis. There are many areas which are worth considering in future reasearch.

Regarding the general segmentation deep learning model architectures, there are several paths to follow. There is the path of exploring other U-Net architectures mentioned in Section 4.3.3 and apply them on the presented datasets in the hope of improved results. There is also a fresh result from this year about the success of the vision transformer models and their connection to the convolutional networks. In [24] they investigate the causes oft the success of the former models compared to the latter ones. They conclude that the differences are mainly coming from micro and macro changes in the architectures and the training methods (like activations, separable convolutions, different grouping of architecture blocks etc.) rather than the difference in the core ideas between the two model family. Inspired by this, I am curious whether the results produced by their novel backbone convolutional network (ConvNeXt) are reproducable on convolutional segmentation networks.

There are ways to continue the reserach regarding the loss functions, as well. The presented results about submodularity only took into consideration the Jaccard loss. To my knowledge, there has not yet been extensive research about submodularity in the context of other losses like Dice or even the genaral Tversky loss. This is an area that can be worth exploring.

Regarding noisy annotations and inbalanced datasets, there is plenty of techniques and approaches to try and study. These areas will play a cricial role in the course of achieving better, medically acceptable predictions for the data of the Korányi dataset.

Appendix

Experiment 1

Median of the validation results from the last 5 epochs are taken in every run. The final result is the mean of these medians over 5 runs. All three U-Net variants were trained with the same hyperparameters shown in Table 5. The only difference was the model architecture. Training took about 1.5-2 hours per run.

In Figure 12 we can see an example of predictions of the different U-Net models. It is clear in this example that all models find the infected area of the left side of the lung. The infected area of the right side seems to be a harder to predict, but the more advanced a model is, the more it can detect of it.

adam beta_1:	0.9
adam beta_2:	0.999
augment:	1
batch norm weight initialisation scheme:	torch default
batch size:	8
batchwise loss:	1
focal gamma:	2
learning rate:	0.0001
loss:	focal Tversky
number of epochs:	30
number of trials:	5
optimizer:	Adam
val split percentage:	0.2
weight decay:	0
weight initialisation scheme:	torch default
weight of false negatives:	0.5
weight of false positives:	0.5

Table 5: Parameter table for the U-Net variants experiments

Experiment 2

Both loss functions were tested with the same hyperparameters and with the same model (shown in Table 6). The presented results are the validation results of the last epoch, average (mean) over 5 runs with different random initialization. Training took about 1.75 hour per run.

In Figure 13 an example prediction is shown with both losses. The left side is more or less correctly detected by both models. On the other hand, the model with the Lovász loss predicts the correct half of the right lung.

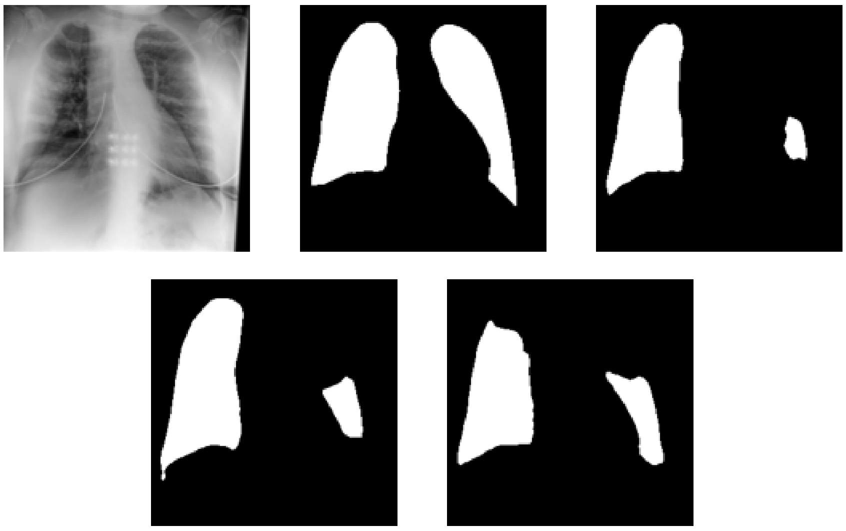
Figure 12: Top (left to right): original image, annotation, U-Net prediction. Bottom (left to right): Att. U-Net prediction, Res. Att. U-Net prediction.

adam beta_1:	0.9
adam beta_2:	0.999
batch norm weight initialisation scheme:	torch default
batch size:	8
dropout after conv:	1
dropout rate:	0.5
learning rate:	0.0001
loss:	Lovász loss /Jaccard loss
model:	Residual Attention U-Net
number of epochs:	30
number of trials:	5
optimizer:	Adam
val split percentage:	0.2
weight decay:	0
weight initialisation fan mode:	fan in
weight initialisation scheme:	He normal

Table 6: Parameter table for Lovász loss vs Jaccard loss experiments

Experiment 3

The hyperparameters of the compared models sometimes differ in this case. Table 7 and 8 show these parameters for Boundary loss combination with Tversky and BCE losses respectively. The results of a run here are the mean of validation scores in the last 3 epochs. Results are from 1 run in every loss setup except pure Tversky and BCE, where the results are the mean of 5 runs. Training took around 1.5-2 hours for each run on one GPU.

One Figure 14 and Figure 15 we can see the predictions of the different models. One interesting thing to mention is that in the case of the Tversky images we can see

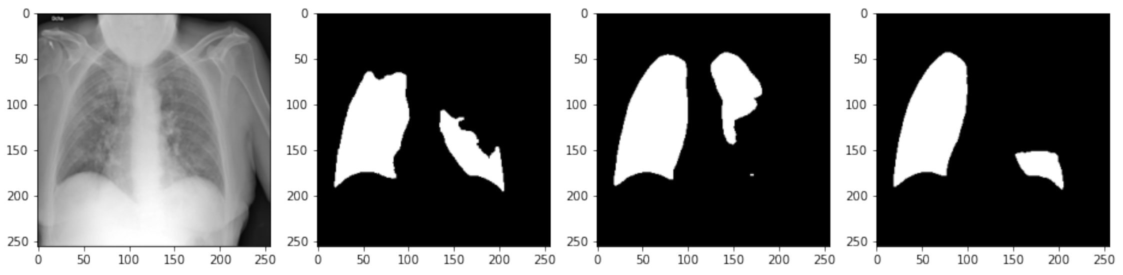

Figure 13: Left to right: original image, annotation, Jaccard loss prediction, Lovász loss prediction.

more smooth borders while in the BCE images the borders are more fragmented. At first we could say that the smooth borders are better, since it resembles the annotation more. But since we have really rough, noisy annotation the BCE images indicate a better generalization. It says that the model actually learned where the border of the tissue is, and it wont predict cancerous area where there is no tissue. This example illustrates the difficulty of evaluating models and predictions on this task and dataset.

	Boundary	**Tversky**	**Combined(constant)**	**Combined(linear)**
adam beta_1:	0.9	0.9	0.9	0.9
adam beta_2:	0.999	0.999	0.999	0.999
batch size:	8	8	8	8
crop size:	256	256	256	256
crop validation images:	1	1	1	1
learning rate:	1.0E-5	0.0001	1.0E-5	1.0E-5
loss:	Boundary	Tversky	Boundary loss: 0.5 Tversky: 0.5	Boundary loss: 0.5 Tversky: 0.5
model:	U-Net	U-Net	U-Net	U-Net
non-uniform sampling:	1	1	1	1
number of epochs:	100	200	100	100
number of trials:	1	5	**1**	**1**
optimizer:	Adam	Adam	Adam	Adam
random pick:	1	1	1	1
rotation limit:	180	180	180	180
sampling method:	positives only	positives only	positives only	positives only
shuffle validation data:	1	1	1	1
val split percentage:	0.2	0.2	0.2	0.2
weight decay:	0	0	0	0
weight of false negatives:		0.7	0.7	0.7
weight of false positives:		0.3	0.3	0.3
distance map bounds:			128	128
distance map rescale factor:			128	128
update rule:			constant	linear rebalance
epochs between loss coef. change:				20
loss coefficient change factor:				0.2

Table 7: Tversky and Boundary experiment parameters

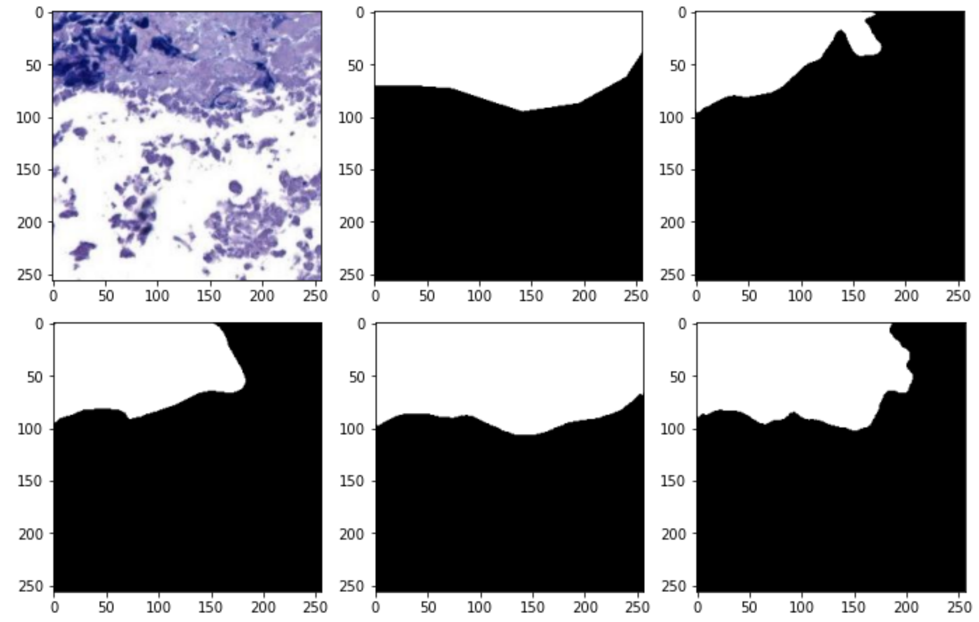

Figure 14: Top (left to right): original image, annotation, Boundary loss prediction. Bottom (left to right): Tversky loss prediction, combined loss (constant) prediction, combined loss (linear) prediction.

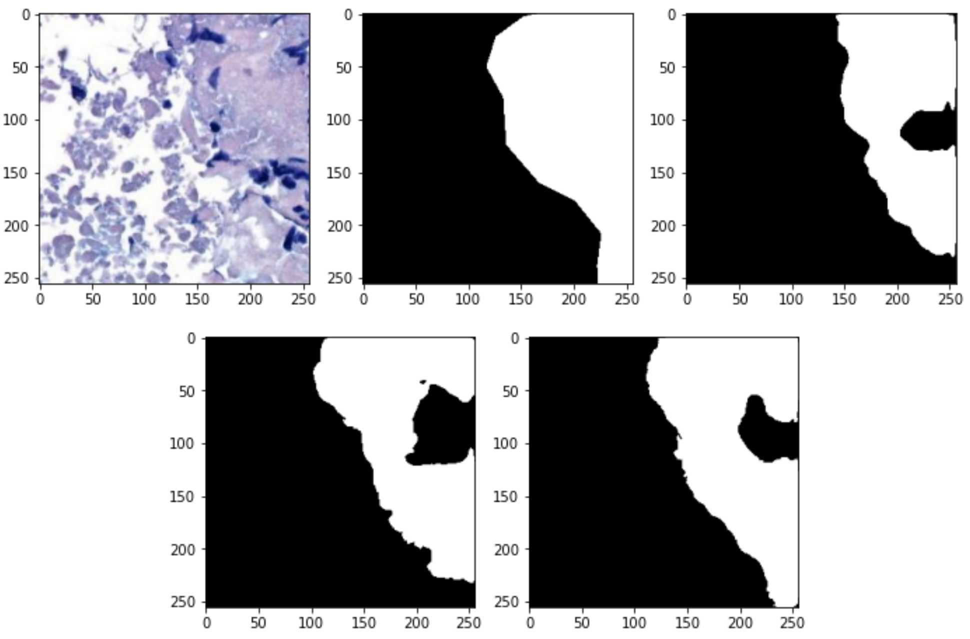

Figure 15: Top (left to right): original image, annotation, Boundary loss prediction. Bottom (left to right): BCE loss prediction, combined loss (constant) prediction.

	Boundary	**BCE**	**Combined (constant)**
adam beta_1:	0.9	0.9	0.9
adam beta_2:	0.999	0.999	0.999
batch size:	8	8	8
crop size:	256	256	256
crop validation images:	1	1	1
learning rate:	1.0E-5	0.0001	1.0E-5
loss:	Boundary	weighted BCE	Boundary loss: 0.5 weighted BCE: 0.5
model:	U-Net	U-Net	U-Net
non-uniform sampling:	1	1	1
number of epochs:	100	100	100
number of trials:	1	5	1
optimizer:	Adam	Adam	Adam
random pick:	1	1	1
rotation limit:	180	180	180
sampling method:	positives only	positives only	positives only
shuffle validation data:	1	1	1
val split percentage:	0.2	0.2	0.2
weight decay:	0	0	0
weight of positive examples:		0.7	0.7
update rule:			constant
distance map rescale factor:			128
distnace map boundas:			128

Table 8: BCE and Boundary experiment parameters

References

[1] 13.10. Transposed Convolution — Dive into Deep Learning 0.17.5 documentation. https://d2l.ai/chapter_computer-vision/transposed-conv.html. (Accessed on 05/19/2022).

[2] Computer Vision: Instance Segmentation with Mask R-CNN | by Renu Khandelwal | Towards Data Science. https://towardsdatascience.com/computer-vision-instance-segmentation-with-mask-r-cnn-7983502fcad1. (Accessed on 12/06/2021).

[3] Faster R-CNN explained for object detection tasks | paperspace blog. https://blog.paperspace.com/faster-r-cnn-explained-object-detection/. (Accessed on 12/06/2021).

[4] Using U-Net deep convolutional neural networks to segment ventricle from MR image (tensorflow). http://jidan.sinkpoint.com/MRI_ventricle_UnetCNN/. (Accessed on 12/06/2021).

[5] Md Zahangir Alom, Chris Yakopcic, Tarek M Taha, and Vijayan K Asari. Nuclei segmentation with recurrent residual convolutional neural networks based U-Net (R2U-Net). In *NAECON 2018-IEEE National Aerospace and Electronics Conference*, pages 228–233. IEEE, 2018.

[6] Anas M. Tahir, Muhammad E. H. Chowdhury, Yazan Qiblawey, Amith Khandakar, Tawsifur Rahman, Serkan Kiranyaz, Uzair Khurshid, Nabil Ibtehaz, Sakib Mahmud, and Maymouna Ezeddin. COVID-QU-Ex Dataset, 2022.

[7] Orhun Utku Aydin, Abdel Aziz Taha, Adam Hilbert, Ahmed A Khalil, Ivana Galinovic, Jochen B Fiebach, Dietmar Frey, and Vince Istvan Madai. On the usage of average Hausdorff distance for segmentation performance assessment: hidden error when used for ranking. *European Radiology Experimental*, 5(1):1–7, 2021.

[8] Francis Bach et al. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends® in Machine Learning*, 6(2-3):145–373, 2013.

[9] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The Lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4413–4421, 2018.

[10] Yuri Boykov, Vladimir Kolmogorov, Daniel Cremers, and Andrew Delong. An integral solution to surface evolution PDEs via geo-cuts. In *European Conference on Computer Vision*, pages 409–422. Springer, 2006.

[11] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[12] Jieneng Chen, Yongyi Lu, Qihang Yu, Xiangde Luo, Ehsan Adeli, Yan Wang, Le Lu, Alan L Yuille, and Yuyin Zhou. TransUNet: Transformers make strong encoders for medical image segmentation. *arXiv preprint arXiv:2102.04306*, 2021.

[13] Xiaocong Chen, Lina Yao, and Yu Zhang. Residual attention u-net for automated multi-class segmentation of covid-19 chest ct images. *arXiv preprint arXiv:2004.05645*, 2020.

[14] Foivos I Diakogiannis, François Waldner, Peter Caccetta, and Chen Wu. ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 162:94–114, 2020.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[16] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[19] Jitesh Jain, Anukriti Singh, Nikita Orlov, Zilong Huang, Jiachen Li, Steven Walton, and Humphrey Shi. SeMask: Semantically masked transformers for semantic segmentation. *arXiv preprint arXiv:2112.12782*, 2021.

[20] Davood Karimi and Septimiu E Salcudean. Reducing the Hausdorff distance in medical image segmentation with convolutional neural networks. *IEEE Transactions on medical imaging*, 39(2):499–513, 2019.

[21] Hoel Kervadec, Jihene Bouchtiba, Christian Desrosiers, Eric Granger, Jose Dolz, and Ismail Ben Ayed. Boundary loss for highly unbalanced segmentation. In *International conference on medical imaging with deep learning*, pages 285–296. PMLR, 2019.

[22] Fei-Fei Li, Justin Johnson, and Serena Yeung. Detection and segmentation. `cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`.

[23] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[24] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[26] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.

[27] Sankaranarayanan Piramanayagam, Eli Saber, Wade Schwartzkopf, and Frederick W Koehler. Supervised classification of multisensor remotely sensed images using a deep learning framework. *Remote Sensing*, 10(9):1429, 2018.

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[29] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[30] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3D fully convolutional deep networks. In *International workshop on machine learning in medical imaging*, pages 379–387. Springer, 2017.

[31] Nahian Siddique, Paheding Sidike, Colin Elkin, and Vijay Devabhaktuni. U-Net and its variants for medical image segmentation: theory and applications. *arXiv preprint arXiv:2011.01118*, 2020.

[32] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7262–7272, 2021.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[34] Jiaqian Yu and Matthew B Blaschko. The Lovász hinge: A novel convex surrogate for submodular losses. *IEEE transactions on pattern analysis and machine intelligence*, 42(3):735–748, 2018.

[35] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6881–6890, 2021.

[36] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 3–11. Springer, 2018.