4 SOLUTION

To sum up, all of the information I mentioned above, the project's primary goal is to simplify medical assistant duties, perform quality and reliable chronic wound segmentation and assist doctors. It is crucial to create a user-friendly tool that is unsophisticated enough, so the ordinary person (in our case medical worker) will quickly understand how to use it. This factor may help to avoid many utilization complications that have been mentioned previously. In addition, it is also essential to provide a proper algorithm that will work out the input picture, so the doctor has no need to make significant corrections manually.

4.1 The Main Concept of the Implementation

In the project, I decided to make a web application and fulfill the most important requirements for diabetic foot photo processing. The software will communicate with the chosen neural network (U-Net or DeepLabV3) and perform segmentation on its best trained model. The main task of the application is to divide distinct tissue types into different classes, and display their ratio on the graph. In practice, such output information is a foundation for future injury medical examination.

4.2 Wound Image Dataset

4.2.1 Data Gathering

Unfortunately, I did not get factual data obtained in Slovak clinics for the examination, so I had to collect them from the open-source databases on the Internet. Consequently, I used some photos from publicly available datasets provided for Foot

Ulcer Segmentation Challenge 2021 (FU-Seg), endorsed by MICCAI[3]. This dataset may not be optimal for the local Slovak needs, so for a better performance network model has to be pretrained.

In the next step, I also made a little preprocessing (generalizing phototypes and their sizes) and deleted unnecessary info (photos with people from another race or non-chronic wounds at all) that are not common to the local site.

4.2.2 Annotation

At first, I annotated all of the pictures into two main segments (wound area and background) to generate a binary mask for each image.

To perform this, I created my own annotation tool based on the graph cut method (see figure 4.1). For proper usage of this tool user has to perform a few simple steps:

- draw a rectangle around the wound area (as close to the injured area as it is possible); the outside region around the wound will be marked as background; the inside one will be segmented similarly but still may be bettered and worked out in the following steps;
- in case the results were not good enough, in this step, the user has to go on and highlight the wound area with a brush in red colors, and not necessary background with the blue one; this step may be performed till the user will not be satisfied with the output results.

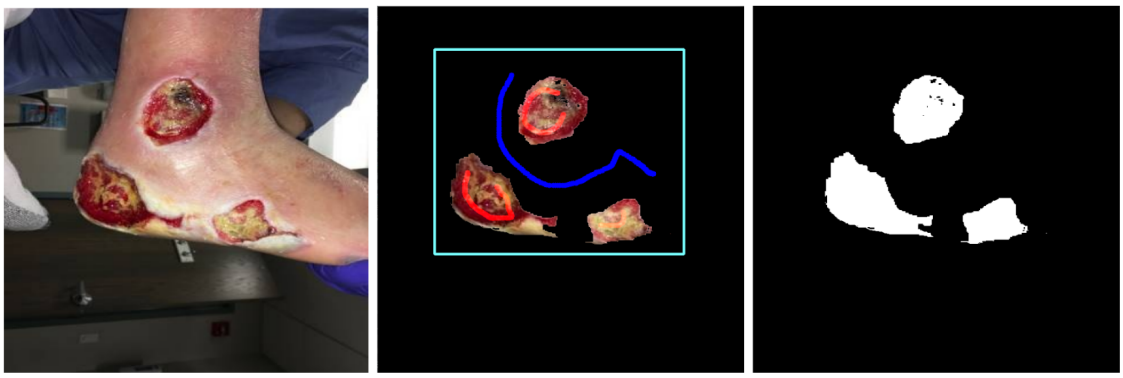

Fig. 4.1: Graph Cut implementation for dataset annotation
from left to right: original image, handwrite marking
of the background and foreground, binary mask

[3]Source link: *https://fusc.grand-challenge.org/FUSeg-2021*

Later I decided to improve my masks and divide photos into more classes. I decided to perform multiclass segmentation. That is why I annotated pictures into granulation (red), slough (green), necrosis (blue), and background (black) classes. Some examples of the masks are displayed in the figure 4.2.

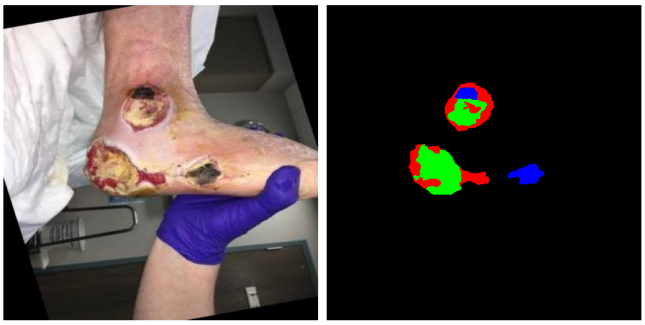

Fig. 4.2: Example of annotated piece of data
(Original Image with its Segmentation Mask)

In addition, I was considering adding the fifth class (the 'neutral' one - in case there is no precise way to estimate the wound; it appears when wound tissue belongs to a few classes or neither of them; for instance, injured visible tissue under the skin surface). However, after the consultation with the doctor, I came to the conclusion that it is pointless, as the injured area counts on the inside damaged contour. In summary, the annotations were checked and filtered by the health worker, so the final dataset is similar to the ground truth.

4.2.3 Dataset Structure

The final wound dataset consists of 350 annotated diabetic foot ulcer photos. Each picture is saved in the .png type of the total size of 512 × 512 pixels. The general class distribution of different wound tissues is shown in the figure 4.3.

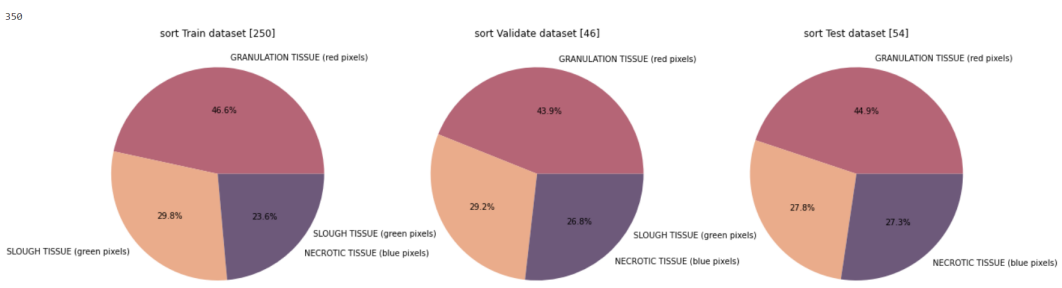

Fig. 4.3: Dataset class distribution

Actually, during my first experiments, I used a smaller dataset with a more significant gap among class ratios (at around 70% of the dataset consisted of the granulation class), which caused a lower accuracy in the output. Therefore, to make the dataset more balanced, more foot photos were added and data augmentation was performed.

In the next step, the whole dataset was divided into three sets: train dataset (70% of the whole amount of data), test and validation datasets (15% for each one).

4.3 CNNs for Image Segmentation

The main algorithm for image segmentation will be discussed in this part of the project.

4.3.1 Demand Specification

First of all, some functional and non-functional requirements have to be established.

Non-functional requirements

- ✧ In each neural network model initialization and training phase has to be performed with the use of reliable open-source frameworks (for example, PyTorch), developer tools for ML (Weights & Biases), and other libraries;
- ✧ A neural network model must be trained, evaluated and tested on the appropriate number and ratio of database samples;
- ✧ Each neural network performance has to be tested and evaluated with some score metrics (localization accuracy, average analyzing time, precision, recall, MCC, IoU, Dice index, etc.);
- ✧ Input wound photo annotations have to be approved by the Slovak health worker;
- ✧ Time spent for analyzing one image must be near to 0.5 sec.

Functional requirements

- ✧ For image segmentation different neural network models have to be compared;
- ✧ Each neural network has to perform multiclass segmentation of four different classes (granulation, slough, necrosis, and background).

4.3.2 Network Models

To perform image segmentation, I selected two neural network models that used to have a good performance in other examinations.

- **U-Net Model**

This type of convolutional network consists of two parts: encoding (contracting) and decoding (expanding) one (see figure 4.4). A model may have great performance efficiency by using augmented data samples only of a few pictures [19].

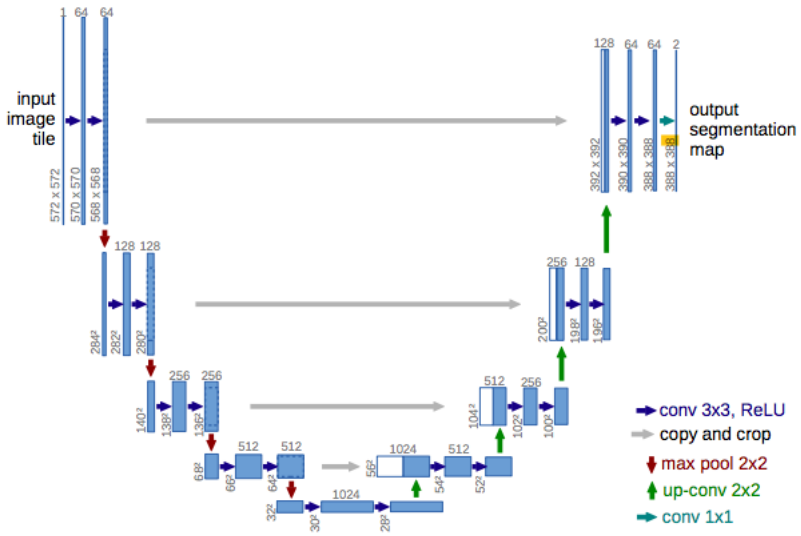

Fig. 4.4: A framework of U-Net neural network model [19]

Input to the model is an RGB wound photo. The first part includes a sequence of two 3 × 3 convolutions, each of which is followed by ReLU function and batch normalization between the convolutions. Then pooling operation is used. Such sequence is applied to the input image four times, which creates the contracting path of the U-Net model. Each time the input image is decreasing in its dimension size, when the number of channels rises.

Afterwards to the current image some equivalent parts of image from the contracting path are added. Moreover, previous sequences of 3 × 3 convolution with the supplementary operations are applied. However, this time 2 × 2 transposed convolution is used to create an expanding path and reduce the number of channels by half. Furthermore, the final convolution and activation function are applied. In this

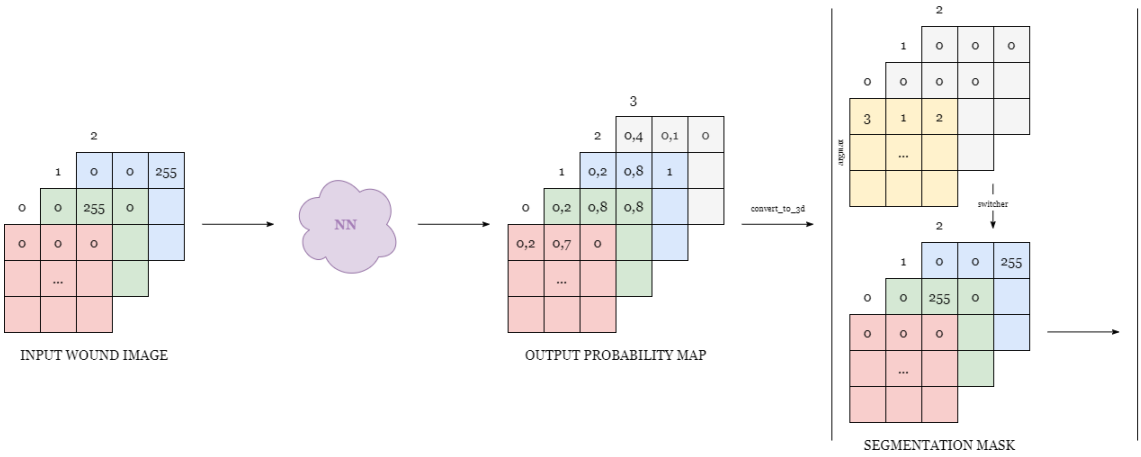

Fig. 4.5: Scheme of U-Net model performance

examination, two final activation functions (ReLU and Sigmoid) and their impact on the outcome have been analyzed. The output of the model is a probability map (512 × 512 pixels) with four channels, where each pixel value (at a range [0; 1]) represents the confidence of belonging to an exact class (see figure 4.5).

- **DeepLabV3 Model**

Another NN model is DeepLabV3[4] - an open-source model presented by Google.

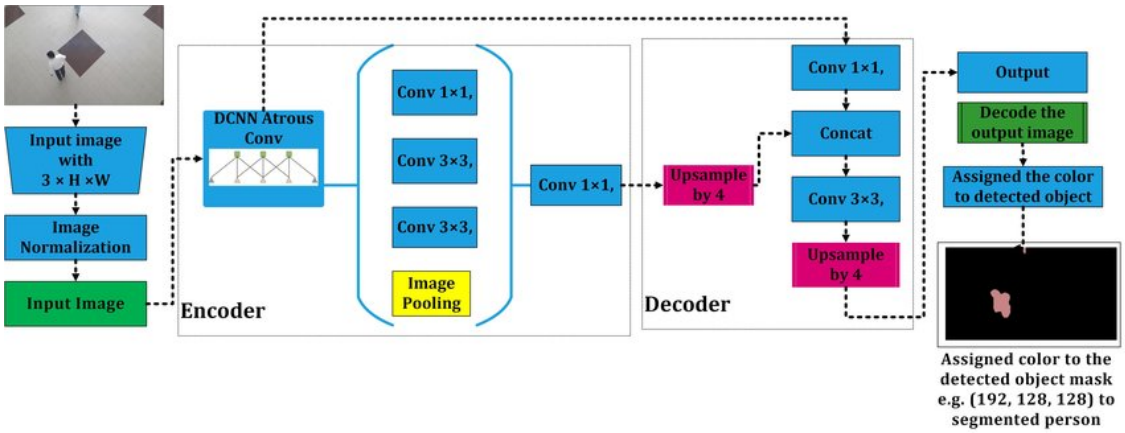

Fig. 4.6: A framework of DeepLabV3 model [20]

Its architecture is displayed in the figure 4.6. In this case, transfer learning has been used to perform wound semantic segmentation. Input to the model is a wound image with three channels for RGB colors. The minimum photo size is 224 × 224 pixels, so I have used an already existing dataset (with each image size 512 × 512 pixels). In addition, the number of output channels has to be set (four for each class) in the

[4]Source link: *https://pytorch.org/hub/pytorch_vision_deeplabv3_resnet101/*

model structure. The same as the previous CNN, DeepLabV3 architecture consists of two parts: encoding and decoding [20]. These parts use a sequence of convolution and pooling operations. The only significant difference is the usage of dilated (or atrous) convolution for the feature extraction. This convolution type is similar to the ordinary one but contains pixel omission (by adding some 'holes' to the kernel window; the number of 'holes' may be set in the l (dilation factor) parameter) and covers a bigger area of the input image. Also, ResNet101 is a backbone for this model and identifies the length of the feature vector (which is 2048) [21]. The output of the model is a dictionary with two tensors. One of them contains loss values for each pixel, while the other is an output probability map. This map consists of unnormalized values, which mark the pixel likelihood of belonging to an exact class.

4.3.3 Training Phase

U-Net model with a Sigmoid activation function I trained on my annotated data with the following parameter values: 1000 number of epochs (650 for a DeepLabV3, as it has longer computation time; due to the limitations I did not have an opportunity to run the whole training on 1000 epochs), the learning rate was 10^{-5}, and a batch window size of 4 (so I would not run out of memory). Every 20 epochs (in case of DeepLabV3 model - every 3), I stored the model weights so, in the future, I would have an opportunity to load a necessary model (the one with the optimal results and without overfitting). The same parameters were used for the U-Net model with a ReLU function and DeepLabV3 training, except the learning rate value, which was $\frac{1}{2} \cdot 10^{-5}$ for the first, and 10^{-6} for the second model. I also used the cross-entropy loss function for all model performance evaluation. Additionally, I added an Adam optimizer to the models, as it is a general rule of thumb. To check model performance (see figure 4.7), I also control validation loss progress. For their tracking, I used the wandb library [22].

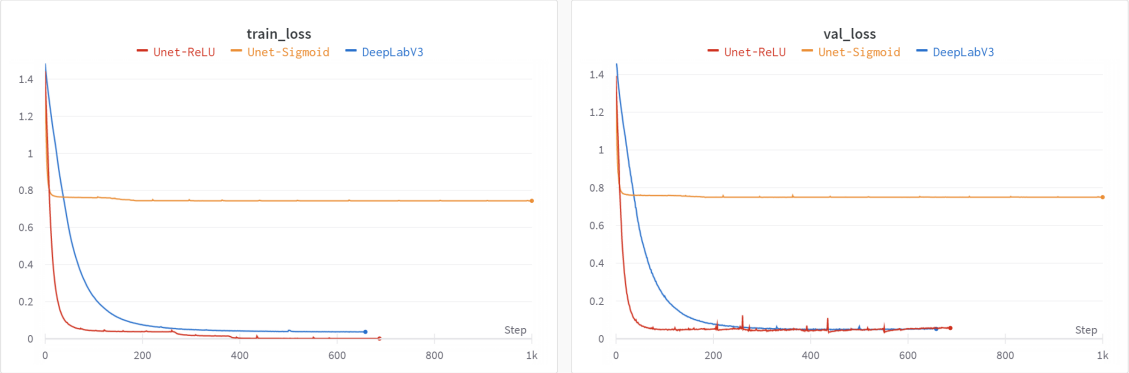

Fig. 4.7: Performance of Cross-entropy loss function for train (on the left) and validation (on the right) datasets
on different U-Net and DeepLabV3 models

4.4 Web Application

4.4.1 Demand Specification

In this paragraph, some key requirements for the web application are described. The same as in the previous section, I divided them into two main groups (functional and non-functional ones).

Non-functional requirements

- ✧ A WIP has to be available for both authenticated and not authenticated users;
- ✧ A WIP has to provide an output in no longer than 10 sec of waiting for single image analysis;
- ✧ A WIP has to be based on the latest up-to-date neural network model;
- ✧ The primary web application has to be functional at the address of the local computer in the computer networking (localhost);
- ✧ A WIP has to be compatible at least with the latest versions of Google Chrome and Microsoft Edge web browsers by May 2022;
- ✧ A WIP has to be developed with the use of reliable open-source frameworks (for instance, Vue.js), extensions, software (npm), and libraries (Chart.js, Flask, and Flask-Cors);
- ✧ A WIP has to provide a clear instruction order to get the output so that the user will easily understand what to do without any additional explanation or learning.

Functional requirements

- ✧ A web page for wound image processing (WIP in the future) has to provide an opportunity to upload at least one photo at a time to make some analysis;
- ✧ A WIP has to provide an opportunity to analyze the input image and generate the output segmentation mask for it;
- ✧ A WIP has to provide an opportunity to download the output mask to the user's personal computer;
- ✧ A WIP has to provide an opportunity to show statistical information about the output with the use of charts;
- ✧ A WIP has to provide an opportunity to reset all of the processes and start the procedure again as many times as the user needs it.

4.4.2 Software Specification

Based on the mentioned requirements, the tool's main purpose is to perform wound photo segmentation on the best chosen U-Net model. The prototype of such a web application is shown in the figure 4.8. Intended audience of the app is medical workers. It is expected for both types of users (the one who already has experience with the tool and the one without it) to have the same time efficiency in the performance.

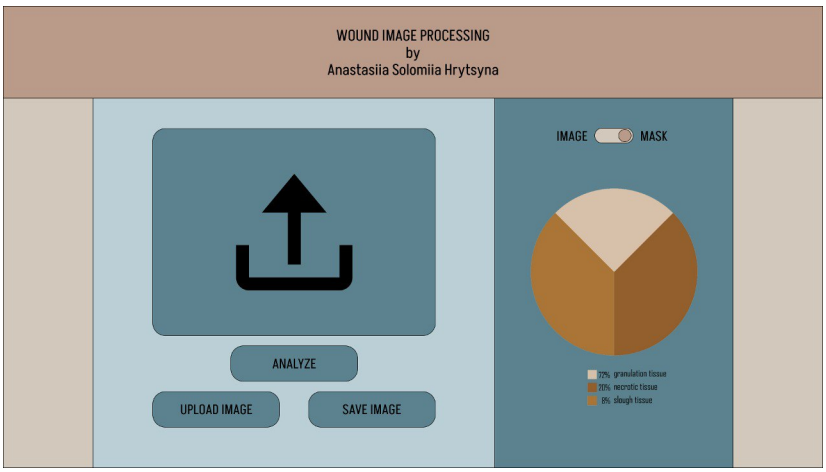

Fig. 4.8: Web application wireframe

4.4.3 User Interface

One of the key states of implemented application is shown in the figure 4.9. Generally, during the execution web page offers the next list of components:

- a title with the name of the work and the author;
- an input place (reserved for the DFU photo upload);
- button ‘analyze’ (to run the script in background and analyze the input);
- button ‘mask toggle’ (to switch between original input and generated mask);
- graph (to show some statistics about the ratio of different mask classes);
- button ‘reset’ (to return to the initial web page);
- button ‘save mask’ (to save a newly created mask to the user’s computer).

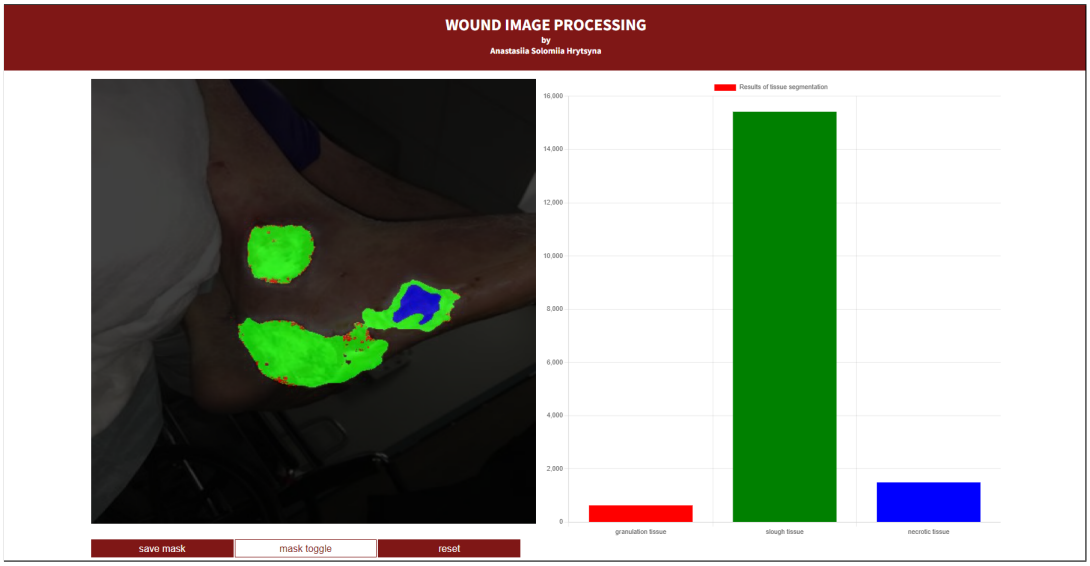

Fig. 4.9: Web application at a final stage of wound segmentation

4.5 Implementation

The web application was implemented using JavaScript (version 1.7) programming language for handling the client-side part, Node.js (version 14.17.6) for the server-side, and Vue.js (version 3.2.33) for building the user interface on the front end.

The initial state software is waiting for input. Entering files must be of the image type (.png, .jpeg, or similar), so the segmentation may be performed. Due to the error handling, any other file type will not be displayed to the user during input selection.

Three new buttons (for image analyzing, mask switching, and reset) occur when the user uploads the necessary image. Image segmentation will not be performed till the user presses the button for analysis. Though, after doing everything right, encoded in base64 image data will be sent through JSON to the route /analyze.

Later, this data is sent as an argument to the program that would be called by the system (analyse-photo.py). In this part Python (version 3.9.5) script handles image preprocessing (decoding to the tensor type, resizing...). Some additional libraries, such as Pillow and Torch, have been used in this step. Next the program loads an already trained U-Net (ReLU) model with the best efficiency (model on the 440th epoch - discussed in the next section) and sends an input image to it. The predicted mask (decoded back to the necessary format - base64) and the number of pixels for each class are returned from the script. Later these values are posted on the website.

After receiving the response, the user will have an opportunity to check a segmented mask by using a button for the mask toggle. In addition, the graph on the right side will show the ratio for each segmented class. Finally, the user may save the output (segmented mask) to their computer. The reset button can be pressed at any application lifecycle stage after the image is loaded. After using it, users have an opportunity to do the whole procedure one more time.

4.6 Evaluation

In this project part general performance of the whole solution will be discussed.

4.6.1 Metrics

For a better and more precise comparison the following metrics were established[5]:

$$\textbf{Precision} = \frac{TP}{TP+FP}$$

$$\textbf{Recall} = \frac{TP}{TP+FN}$$

$$\textbf{F1-score} = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall} = \frac{2 \cdot TP}{2 \cdot TP+FP+FN}$$

$$\textbf{IoU(Intersection over Union or Jaccard Index)} = \frac{Area\,of\,Overlap}{Area\,of\,Union}$$

$$\textbf{Dice score} = \frac{2 \cdot TP}{(TP+FP)+(TP+FN)}$$

$$\textbf{MCC(Matthews' correlation coefficient)} = \frac{(TP \cdot TN)-(FP \cdot FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

[5] **TP** is the equivalent for a true positive value, **TN** - true negative, **FP** - false positive, and **FN** - false negative.

Localization exactness = $\frac{1-binary\ ground\ truth \oplus binary\ nn\ output}{picture\ size}$ (an average percentage of pixels counted for each photo that represent ratio between truly segmented wound area (without no class difference) and the whole picture);

Average performance time - an average time needed for segmenting one picture.

4.6.2 Results

This section will discuss some performance results of the testing set.

Finally, I selected the following models:

- U-Net with ReLU activation function on its 440 epoch (I found out that at nearly the 450th epoch, the validation loss starts to grow slightly, which means the model starts pretraining and overfitting (see figure 4.7)). The total model weight is 29.6 MB.
- U-Net with Sigmoid activation function on its last 1000 epoch (as during all training time, the loss curve remains constant(see figure 4.7)). The total model weight is 29.6 MB too.
- DeepLabV3 on its 630 epoch (similar to the previous example, the loss curve remains constant (see figure 4.7), so I used the last saved model). The total model weight is 233 MB.

	Average			Median			Total Sum		
	MCC	IoU	Dice	MCC	IoU	Dice	MCC	IoU	Dice
DeepLabV3	**0.854**	**0.740**	**0.804**	**0.890**	**0.801**	**0.872**	**0.868**	**0.760**	**0.798**
Unet (Sigmoid)	0.805	0.699	0.754	0.883	0.784	0.861	0.511	0.302	0.313
Unet (ReLU)	0.783	0.653	0.721	0.858	0.744	0.823	0.802	0.657	0.725

Table 4.1: Models evaluation using MCC, IoU and Dice score

The average performance time of each U-Net model is 0.015 sec, while the DeepLabV3 performance takes 10 times longer (0.15 sec). Execution time and model weights are bigger in DeepLabV3 architecture as this model is much deeper than the U-Net one. An average performance time on the web page is 8 sec.

For all of the models, localization exactness is more than 99%, which means that most wound area is segmented successfully compared to the whole picture.

Other counted outcomes are shown in the table 4.1. Average values have been counted for every picture separately, and then the mean value has been counted and represented in the table. The same principle has been used in the 'Median' column, except the final value is the median. The last column contains values counted for the whole set of testing pictures. The best performance shows the DeepLabV3 model (see table 4.1).

		Precision	Recall	F1-score
DeepLabV3	*granulation tissue*	**0.855**	0.823	**0.839**
	slough tissue	0.763	**0.706**	**0.734**
	necrotic tissue	0.893	**0.864**	**0.878**
	background	**0.997**	**0.999**	**0.998**
Unet - Sigmoid	*granulation tissue*	0.204	**0.870**	0.331
	slough tissue	**0.868**	0.611	0.717
	necrotic tissue	0.914	0.693	0.789
	background	0.995	0.961	0.978
Unet - ReLU	*granulation tissue*	0.794	0.831	0.813
	slough tissue	0.729	0.564	0.636
	necrotic tissue	**0.926**	0.641	0.757
	background	0.995	0.998	0.996

Table 4.2: Models evaluation of Precision, Recall and F1-score for different classes separately

The next results are represented in the table 4.2. For most of the scores, DeepLabV3 achieves the best outcomes. The U-Net model with the ReLU function seems to have a more stable outcome performance (without any drops to 30% as with the Sigmoid function).

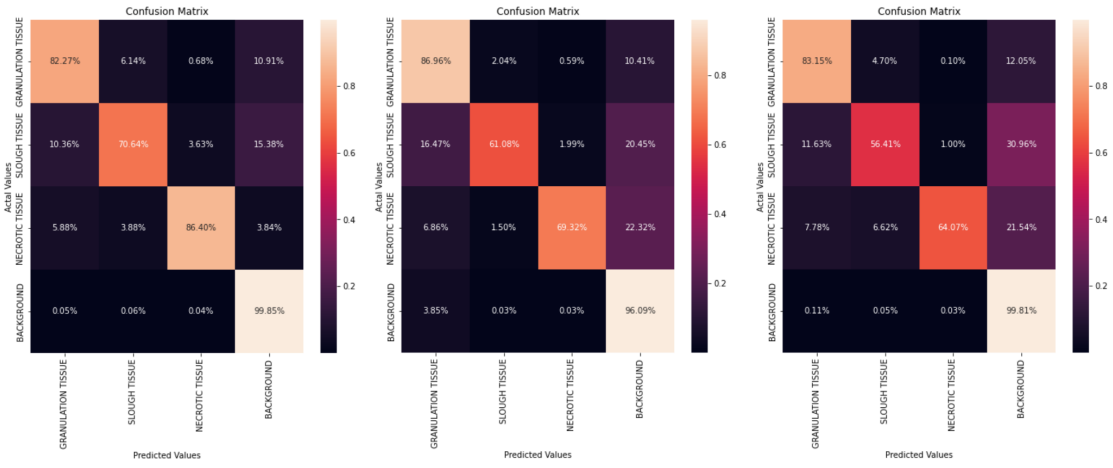

Fig. 4.10: Confusion matrixes from left to right: DeepLabV3, U-Net(ReLU),and U-Net(Sigmoid) models

The confusion matrix for all of the models is displayed in figure 4.10. It shows that usually, all tissue classes are recognized correctly. The most common error is marking wound tissue as a background (not another tissue class). The most challenging class to identify is the slough.

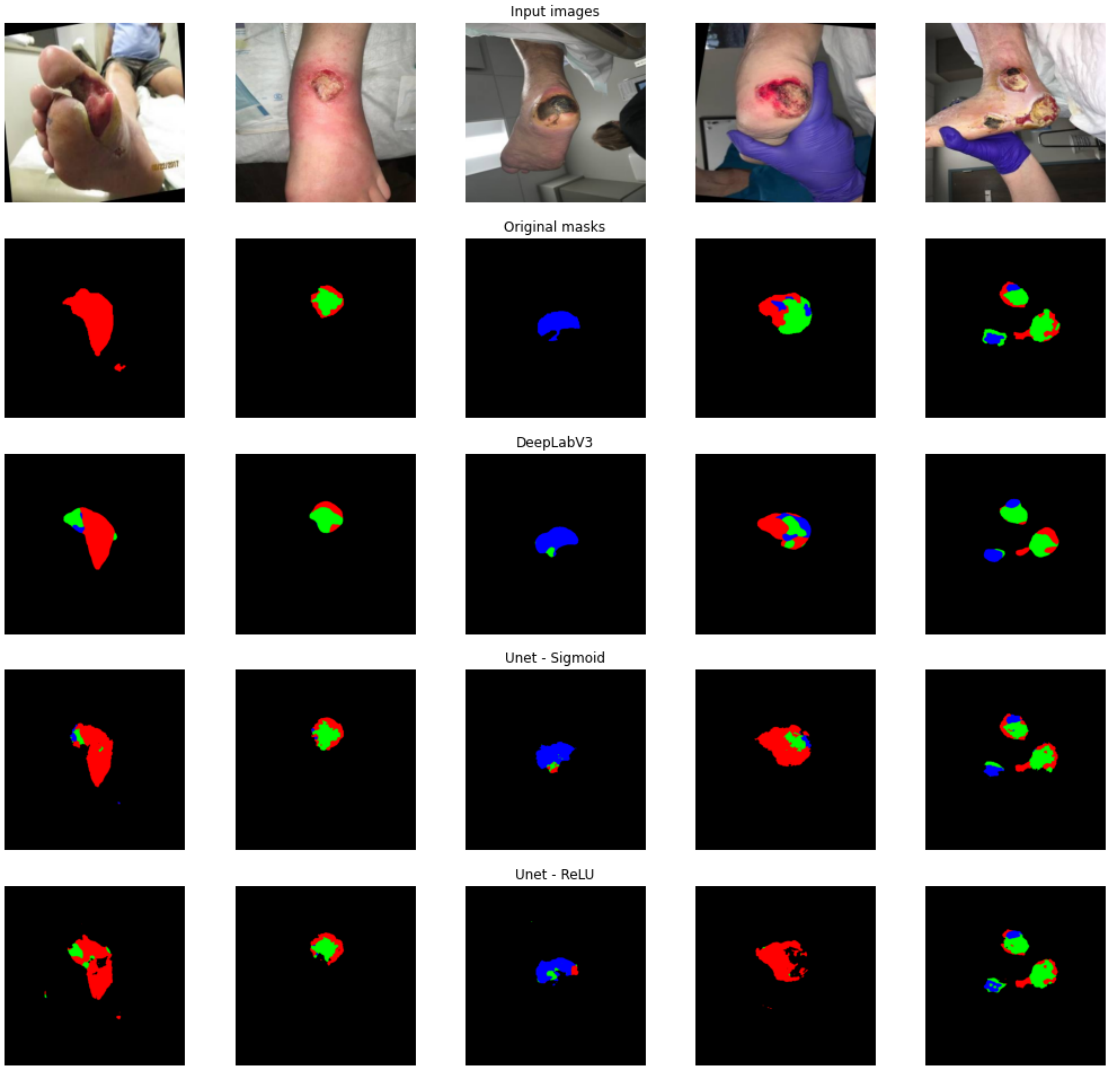

Fig. 4.11: Example of segmentation results for different NN models

An example of wound photo segmentation on different NN models is shown in the figure 4.11.

APPENDIX C: Digital Medium

Registration number: FIIT-5212-105323

Digital medium name: BP_AnastasiiaSolomiiaHrytsyna.zip

File medium consists of three folders containing multiple files:

- Datasets, models, weights
  - 350pics.zip - Dataset containing all the 350 images
  - model_weights.zip - folder containing trained weights for all three models
  - unet_relu_model.zip - python file defining the Unet with ReLU model
  - unet_sigmoid_model.zip - python file defining the Unet with Sigmoid model
  - wound_dataset_splitedlist.zip - folder with .csv files containing lists of images for given dataset (test, train, validate)
- Web application
  - README.md - file containing the main information about web application, its lifecycle, and a tutorial how to start using created software
  - py - folder with the main containing of web application
- Notebooks
  - dataset-eda.ipynb - Kaggle jupyter notebook containing Exploratory Data Analysis of out dataset
  - model_training.ipynb - Kaggle jupyter notebook used for training the models
  - model-evaluation.ipynb - Kaggle jupyter notebook with evaluation metrics and functions used for results comparison between models

All the files included in the folder "Datasets, models, weights" must be uploaded as datasets into each Kaggle notebook. Before running the jupyter notebooks, add apropriate paths for models, datasets and weights. Additionally, read Web application/README.md file to start running web application.