Diabetic retinopathy segmentation

Petr Stádník

May 2023

FACULTY OF ELECTRICAL ENGINEERING
Czech Technical University in Prague

Supervisor: prof. Dr. Ing. Jan Kybic

Acknowledgement

I would like to thank Professor Jan Kybic for leading the project, his advice and patience.

Declaration

I declare that I have prepared the submitted work separately and that I have listed all the information sources used in accordance with Methodological instructions on the observance of ethical principles during preparation this project.

Prague 31. 5. 2023

Abstract

Diabetic retinopathy is the most frequently occurring complication of diabetes mellitus and remains a leading cause of vision loss globally[1]. It consists of damage of the blood vessels in the retina. This imparment mainly affects the working-age population[2]. Actual research is focused to identify better and cheaper ways of identification, management, diagnosis and treatment of retinal disease. Computer-aided disease diagnosis in retinal image analysis could ease mass screening of populations with diabetes mellitus and help clinicians in utilizing their time more efficiently[2]. One of the first tasks in computer diagnose is process retinal images and correctly found affected regions. For this purpose I am going to use image segmentation through neural networks.

Aim

Aim of this project is to develop and test an algorithm for segmentation of retinal lesions associated with Diabetic retinopathy. Using one of the challenges provided by Grand Challenge platform [3], specifically Diabetic Retinopathy: Segmentation and Grading Challenge [4], I would like to get acquainted with the problematic of semantic segmentation in medical images using convolutional neural networks. First I need to download and get to know the dataset, and learn how to access it and visualise it. For these purposes I am going to use Python programming language and I will have to familiarise myself with PyTorch library and tool Conda for package managing. Also I should get familiar with the state of the art methods for medical image segmentation in general and for segmenting diabetic retinopathy lesions in particular. For this purpose I will need to learn basics of convolutional neural networks and deep learning. Result of this project should be a description of the state of the art and identify suitable methods for further experiments. Create implementation or be acquainted with existing implementation of the chosen methods, evaluate their performance experimentally on the IDRiD (Indian Diabetic Retinopathy Image Dataset) [2] and compare my results to the results reported by the challenge participants. Evaluate the dependence on the most important parameters. Due to this project I am also learning, how to correctly use LaTeX at all and BibTeX citations.

Contents

1 Introduction

Diabetic Retinopathy is the most prevalent cause of avoidable vision impairment, mainly affecting the working-age population in the world. [4] In the figures below we can see which differences this disease makes in the eye. In this paper I am going to get to know basics of deep learning methods for image segmentation to better understood how can modern methods helps society to tackle with these problems.

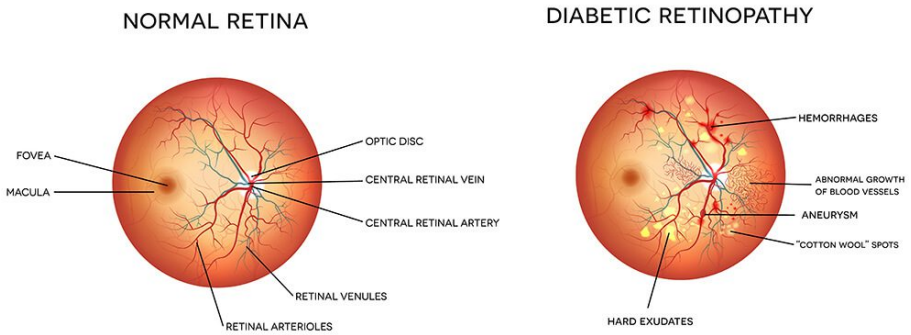


Figure 1: In this picture from Bennett Eye Institute[5] we can see theoretical difference between healthy eye on the left side and eye affected by diabetic retinopathy on the right side.

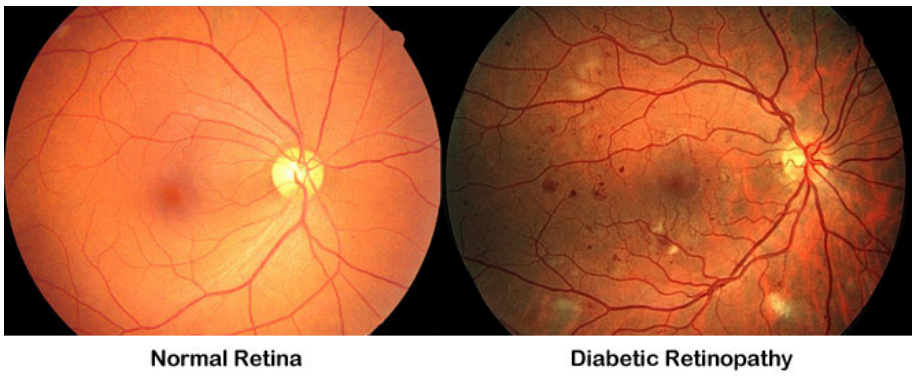


Figure 2: In this picture from Vision Gallery[6] we can see real difference between healthy eye on the left side and eye affected by diabetic retinopathy on the right side.

2 Indian Diabetic Retinopathy Image Dataset (IDRiD)

2.1 Basic information

This dataset represents database for diabecic retinopathy screening reseach. The fundus images in IDRiD database were acquired from an Eye Clinic located in Nanded, (M.S.), India. Retinal images of humans affected by diabetes were captured with 39 mm distance between lenses and examined eye using non-invasive fundus camera having xenon flash lamp. The images have resolution of 4288×2848 pixels and are stored in jpg file format. The size of each image is about 800 KB. The dataset is formed by extracting 516 images from the thousands of examinations done during the period 2009–2017. Experts verified that all images are of adequate quality, clinically relevant and that no image is duplicated and that a reasonable mixture of disease stratification representative of diabetic retinopathy and diabetic macular edema is present. [2]

2.2 Annotation

This data set provides pixel level annotation. For segmentation I use groundtruth images, which are provided separately for each class, as shown in the picture below 3, in tif file format.

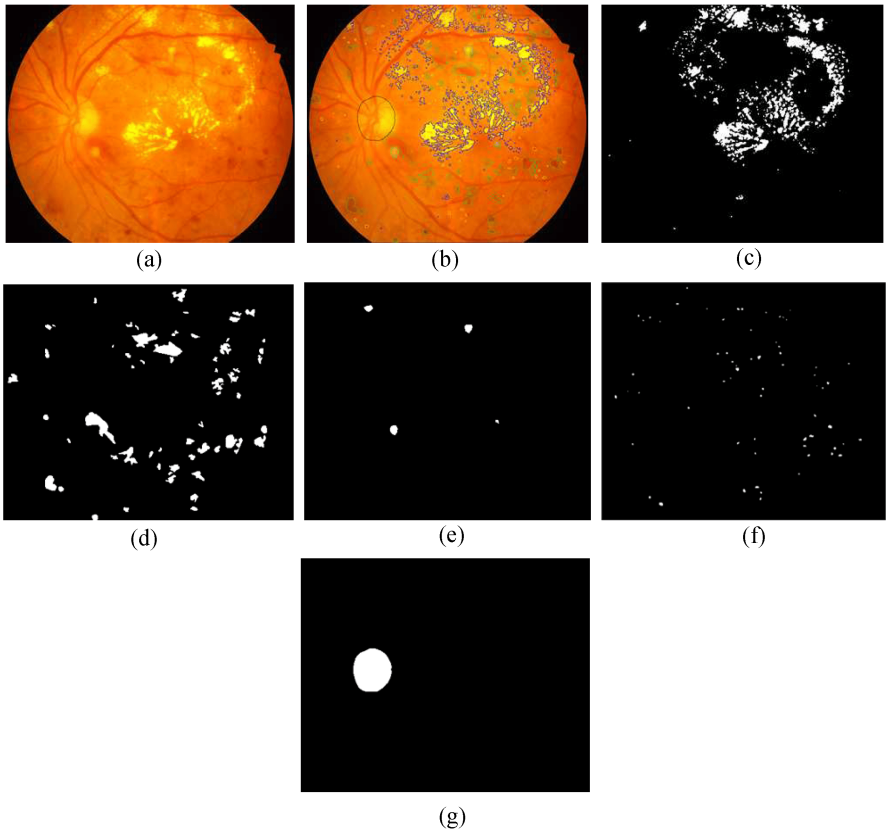


Figure 3: In the picture from dataset description above [2] we can see examples of image's groundtruth contained by dataset. In the picture (a) we can see image sample how it is without any interference. In the picture (b) is the same picture but with markups created by annotator. In pictures (c)-(g) we can see examaple of groundtruth for each class in the following order: hard exudates, hemorrhages, soft exudates, microaneurysms and optic disc.

2.3 Read dataset

I order to read dataset I have created a new class in python, using PyTorch library. Actual code you can find in my GitHub repository. [7]

```
class MyIDRiDImageDataset(Dataset):
    def __init__(self, img_dir, labels_img_dir, normalize=False, resize=None):
        self.labels_img_dir = labels_img_dir
        self.img_dir = img_dir
        self.all_images = os.listdir(self.img_dir)
        self.all_labels = os.listdir(self.labels_img_dir)
        self.normalize = normalize
        self.resize = resize

    def __len__(self):
        return len(self.all_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.all_images[idx])
        image = read_image(img_path).float()
        imlabel = Image.open(os.path.join(self.labels_img_dir, self.all_labels[idx
    ]))
        imlabel = ToTensor()(imlabel)
        imlabel.float()
        transform = transforms.Compose(
            [transforms.Normalize((0.0, 0.0, 0.0), (255, 255, 255))])
        if self.normalize:
            for i in range(3):
                image[i, :, :] = (image[i, :, :] - image[i,:,:].min()) / (torch.max
    (image[i, :,:]) - torch.min(image[i, :, :]))
        else:
            image = transform(image)

        if self.resize:
```

```
            image = transforms.Resize(self.resize, antialias=True)(image)
            imlabel = transforms.Resize(self.resize, antialias=True)(imlabel)

        if (imlabel.max() - imlabel.min()) > 0:
            imlabel = transforms.Normalize(0, (imlabel.max() - imlabel.min()))(
    imlabel)

        imlabel = torch.round(imlabel)

        return image, imlabel
```

Listing 1: Implementation of dataset reader.

3 Data augmentation

3.1 Introduction

Common problem with medical data is that we do not have enough samples in dataset. It is usually caused by high cost of these samples. It is much more easier to take a pictures of cars. We can also hire many people to annotate such images, because you do not have to be expert to recognise a car. In the opposite for example get picture of eye requires to have sophisticated equipment and, also you need enough patients with this particular disease and finally you need real experts to annotate these images, how are usually busy and their time is expensive.

For these reasons there are methods how to artificially extent dataset and make it more robust in order to improve training of network and achieve better results.

3.2 Augmentation methods

In PyTorch, we can use torchvision.transforms to implement augmentation. [8]

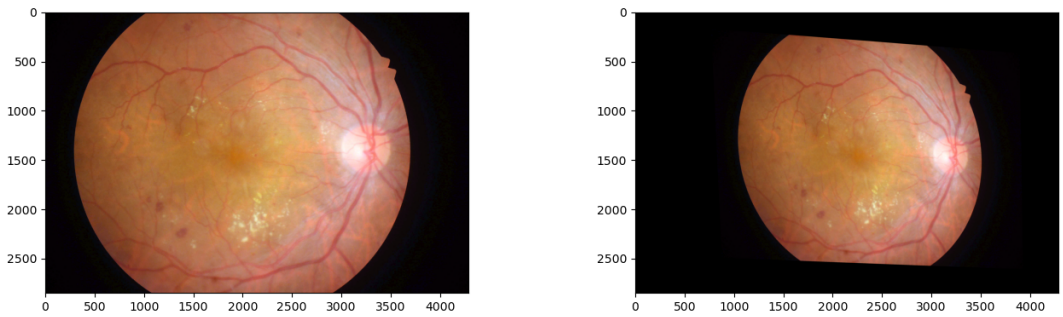
```
import torchvision.transforms
```

Listing 2: Import necessary package

3.2.1 Random perspective

```
transformer = transforms.RandomPerspective(distortion_scale=0.6, p=1.0)
transformed_image = transformer(original_image)
```

Listing 3: Implementation



(a) Original image

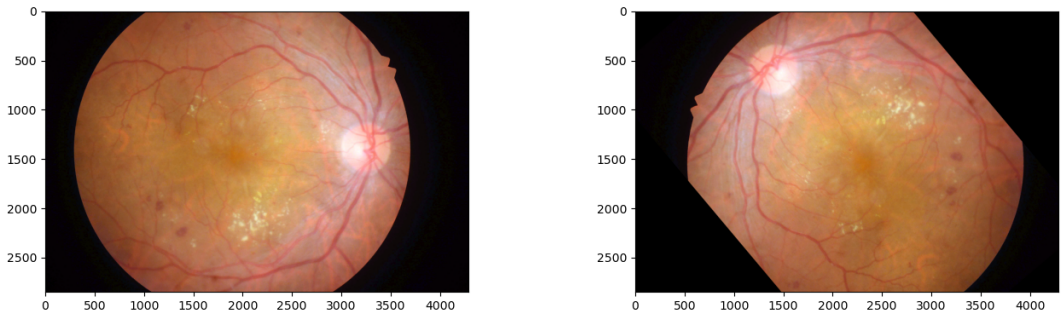
(b) Image with changed perspective

Figure 4: Changing perspective

3.2.2 Random rotation

```
transformer = transforms.RandomRotation(degrees=(0, 180))
transformed_image = transformer(original_image)
```

Listing 4: Implementation



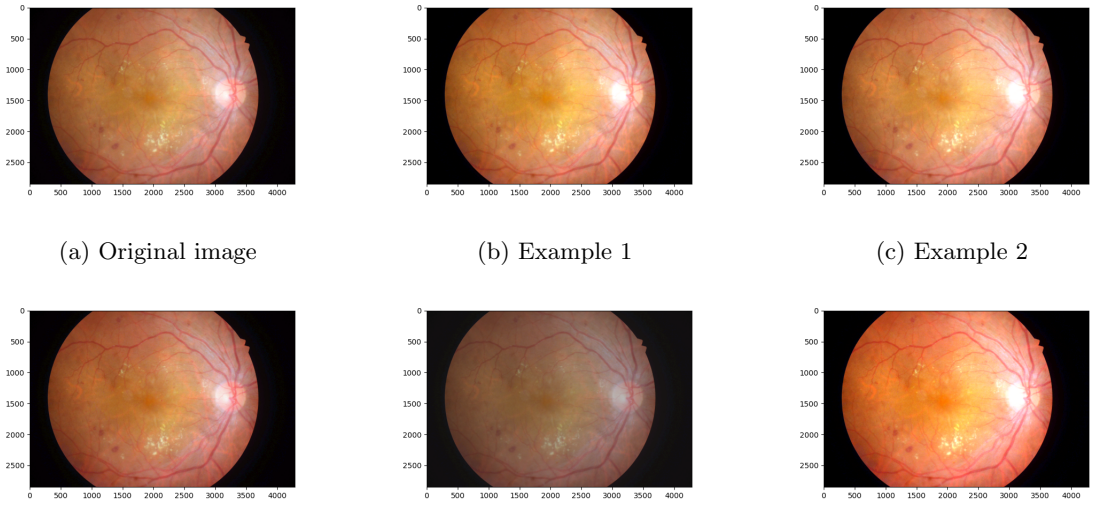
(a) Original image (b) Rotated image

Figure 5: Rotation

3.2.3 Brightness and colours adjustment

```
transformer = transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,
    hue=.02)
transformed_image = transformer(original_image)
```

Listing 5: Implementation



(a) Original image (b) Example 1 (c) Example 2

(d) Example 3 (e) Example 4 (f) Example 5

Figure 6: Examples of random brightness, contrast, saturation and hue adjustments.

3.2.4 Others

There is practically unlimited number of augmentation methods. In sections above I have chosen only few examples. Other methods which we can use are horizontal and vertical flipping, random cropping, adding some noise or use for example Gaussian blur. In practical application we will mostly use combination of methods listed above.

4 Convolutional neural networks

The term Deep Learning or Deep Neural Network refers to Artificial Neural Networks (ANN) with multi layers. [9] One of the most popular deep neural networks is the Convolutional Neural Network (CNN). It is named after mathematical linear operation between matrixes called convolution. CNN has multiple layers including convolutional layer, non-linear layer, pooling layer and fully-connected layer.

The CNN is mostly used in machine learning problems. Specially they seem to be great for the applications that deal with image data, such as image classification and computer vision. Also for in natural language processing (NLP) and finally for image segmentation tasks.

4.1 Brief history

Interesting fact is that they were really derived and inspired by behaviour of neurons in our brains. As shown in experiment with a cat in the figure 11 made by David H. Hubel and Torsten Wiesel in 1959. [10] Especially the way how neurons in brain processes image. In this experiment they have shown up that single neuron does not recognise whole subject, but is sensitive to border of subject in one specific place in field of vision. And we have something like matrix of these neurons to cover our whole field of vision. Actually it make sense, because we want to recognise what we see independently on the position of the object in our field of vision. That is why these convolutional neural networks are so good at image processing and why it gives more sense to use them in comparison to fully connected neural networks.

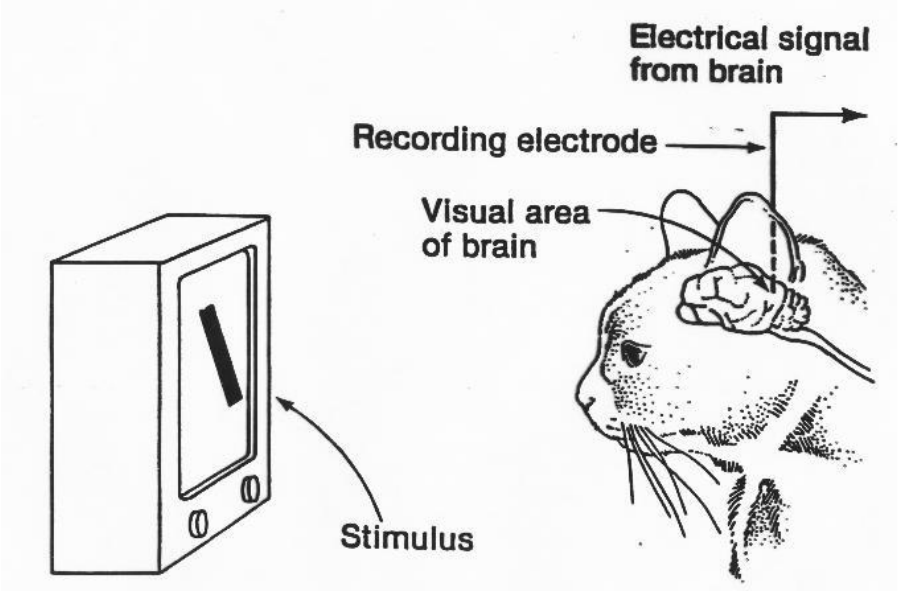


Figure 7: Description of experiment. There was a cat, which had been anaesthetised before experiment. Hubel and Wisel implemented tiny electrode to the one simple neuron in the cat's striate cortex. Before cat's eyes they displayed different images and shapes and recorded reaction of the neuron.

4.2 Convolution

Convolution it selves means mathematical operation of two functions f and g, one of these functions is typically called kernel function, that produces third function $(f * g)$ that expresses how shape of one function is modified by the other one - example in figure 8.

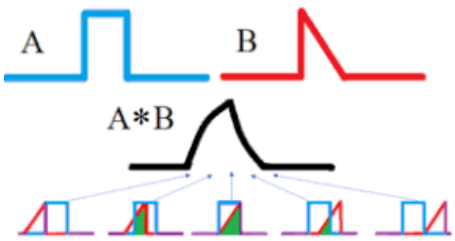


Figure 8: Example of two functions and product of their convolution.

Convolution is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x) \cdot g(t - x)\, dx \tag{1}$$

4.2.1 Discrete 2D convolution

In 2-dimensional discrete case we can define convolution as follow:

$$(F * G)[s,t] = \sum_x \sum_y F[x][y] \cdot G[s-x][t-y] \tag{2}$$

Visually it looks like as we can see in the figure 9.

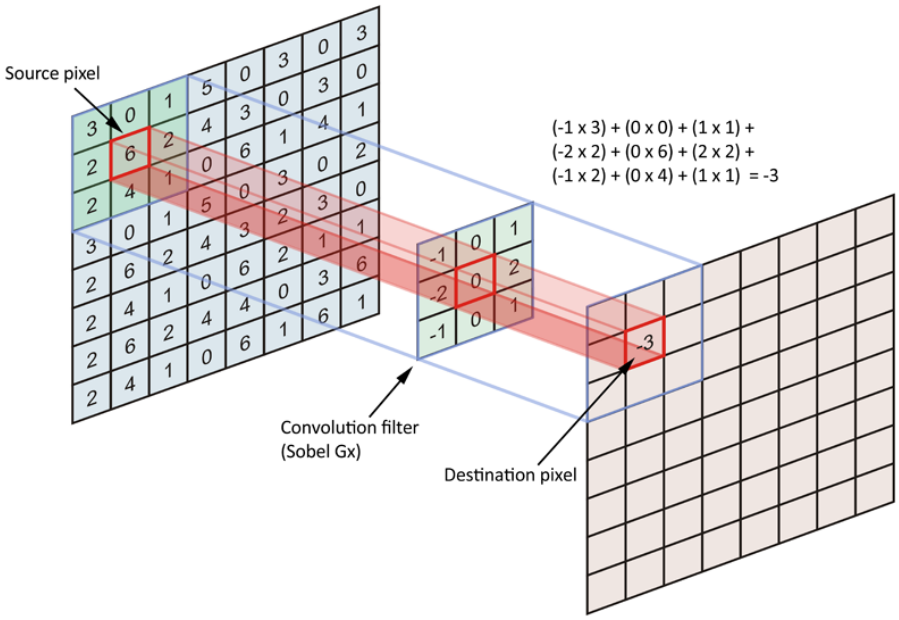


Figure 9: Visualisation of convolution procedure.

4.3 Layers

4.3.1 Convolution layer

The princip of convolution layer is based on convolution described in the section above. Input of this layer is the image on which is proceeded convolution.

In this layer we can set parameters like kernel size - means height and width of the window. Then dilatation, which adds spaces between taken pixel from input image as shown in the figure 10, default value is 1. Also we can set stride parameter which defines for how many pixel will be window moved in each step, default value is 1. In PyTorch implementation of this layer we can also set padding, because naturally result of convolution has smaller size that input image. This relation between input and output image can be calculated as follows.

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

where H_{out} and W_{out} respective H_{in} and W_{in} represents output respective input image height and width.

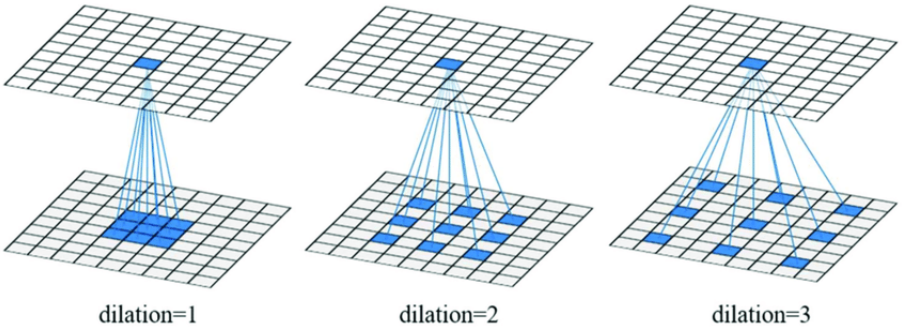


Figure 10: Visualisation of convolution with dilatation parameter set on 1, 2 and 3.

4.3.2 Pooling layer

The pooling layer is used for down-sampling the image in order to reduce the complexity for further layers. It is usually used after at least one convolutional layer. One of the most used variant is Max Pooling, which works as follows.

Procedure is practically similar to convolution as it is described in section above, but we do not calculate any new values, but we just chose maximal value from the values inside the window. That window can also be called kernel. Kernel size is one of the parameter we set in this layer. Another one is stride.

Stride defines for how many pixels we move the window in each step. Stride is usually equal to kernel size.

Another parameter which we can set is dilatation. Dilatation is a parameter that controls the stride of elements in the window.

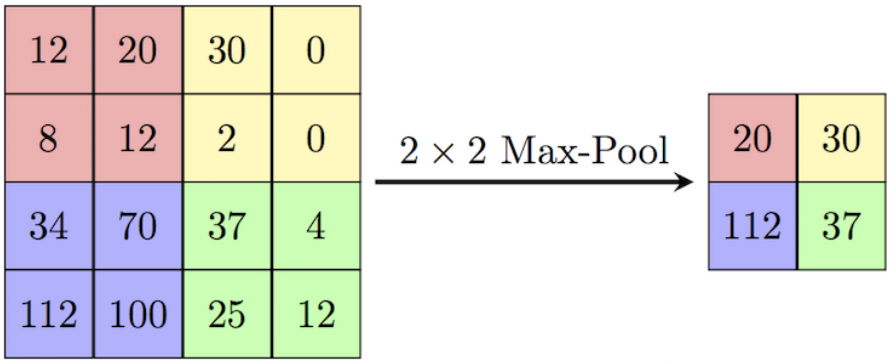


Figure 11: In the picture we can see demonstration of max pooling layer. On the left side is placed input image with size 4 x 4, and numbers in squares represent its pixel values. On this input image we have applied max pooling layer with kernel size 2 x 2, stride 2 and dilatation 1. Output image we can see on the right. Output image has size 2 x 2 and contains max values from each area where window was placed.

4.4 Activation functions

Principe of activation functions is to decide if a neuron/node should be activated or not and introduce some non-linearity into the output of a neuron. There is really a lot of activation functions and their modifications. Below I am going to list just two most used of them.

4.4.1 ReLU

Rectified Linear Unit (ReLU) has following definition.

$$f(x) = max(0, x) \tag{3}$$

practically it means, that returned value will never be negative. It will always returns, 0 for negative inputs and same as input value for positive ones. This function commonly follows convolution layer.

4.4.2 Softmax

Softmax function scales values in range from 0 to 1. It is useful for normalizing of outputs.

This function is usually placed as a last layer of the network, also therefore we can use its output as propability values.

Softmax function is defined as:

$$Softmax(x)_i = \frac{\exp x_i}{\sum_{j=1}^{n} \exp y_j} \tag{4}$$

where x an input vector consist of n elements for n classes. $\sum_{j=1}^{n} \exp y_j$ is a normalisation factor which ensure, that output will be just between 0 and 1.

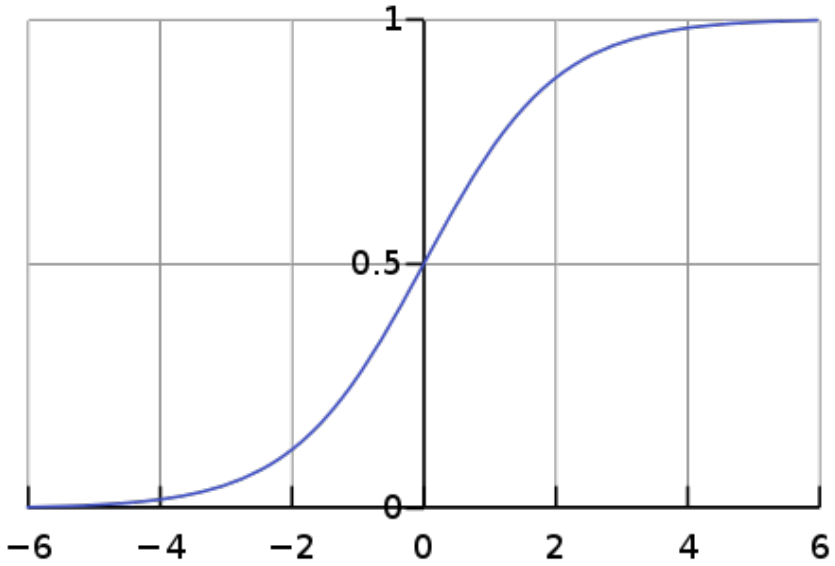


Figure 12: Softmax function.

One important problem connected with using this function is called Vanishing gradient problem. It occurs when x value is far from zero and slope of the function is rally small - means also gradient in this point is rally small, can be almost zero, so training procedure can get stuck.

4.5 Loss functions

Important part for proper learning of neural network is to choose suitable loss function. Loss function is placed in the end of the network, compares results of network with correct results and measure how different they are. The way how it measure differences it is what we call loss function. Our aim or lets say aim of our network is minimise the lost function.

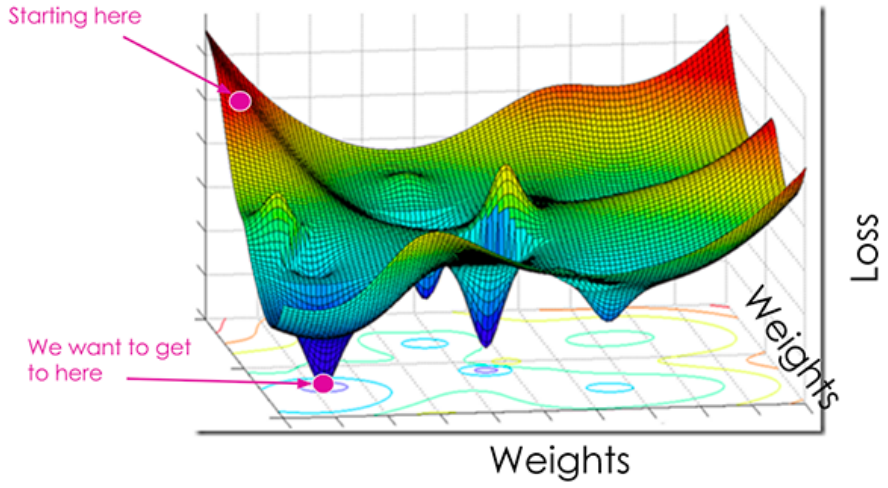


Figure 13: Visualisation of some loss function value according to trained model weights.

4.5.1 Sum of Squared Errors Loss

Sum of squared errors (SEE) is one of the simplest loss function. Formula for SEE is:

$$SSE = \sum_{x} (y - f(x))^2 \tag{5}$$

where y is correct result and $f(x)$ is network result.

4.5.2 Cross Entropy Loss

Cross entrophy loss function is one of the most used loss function in image segmentation tasks. It needs propability values as input, it means values scaled between 0 and 1. Therefore we usually insert softmax function before.

Cross entropy H is defined like:

$$H(p, q) = -\sum_{x} p(x) \cdot \ln q(p) \tag{6}$$

where $p(x)$ is true probability distribution and $q(x)$ is my model's probability distribution.

4.5.3 Dice Loss

Dice loss originates from Sørensen–Dice coefficient (DC) which is statistical measurement of similarity between two samples. Practically it is based on overlap between two measured sets. In formula we can take p like pixels in network result and q like pixels from groundtruth image. Than DC equals to:

$$DC = \frac{2 \cdot \sum_{p,q} p \cdot q}{\sum_{q} p^2 + \sum_{p} q^2} \tag{7}$$

can be also formulated as:

$$DC(X, Y) = \frac{2 \cdot X \cap Y}{|X| + |Y|} \tag{8}$$

from that Dice loss function comes to be

$$DiceLoss = 1 - DC \tag{9}$$

to can be minimalized.

5 Experiments

All my code used in this section you can find here in my GitHub repository. [7]

5.1 Description

For my tests I found already created network model of U-NET type. [11] This type is commonly used for image segmentation tasks. Illustration of this model you can see in the figure 14. This model was primary created for brain MRI segmentation where they used image of size 256 x256.

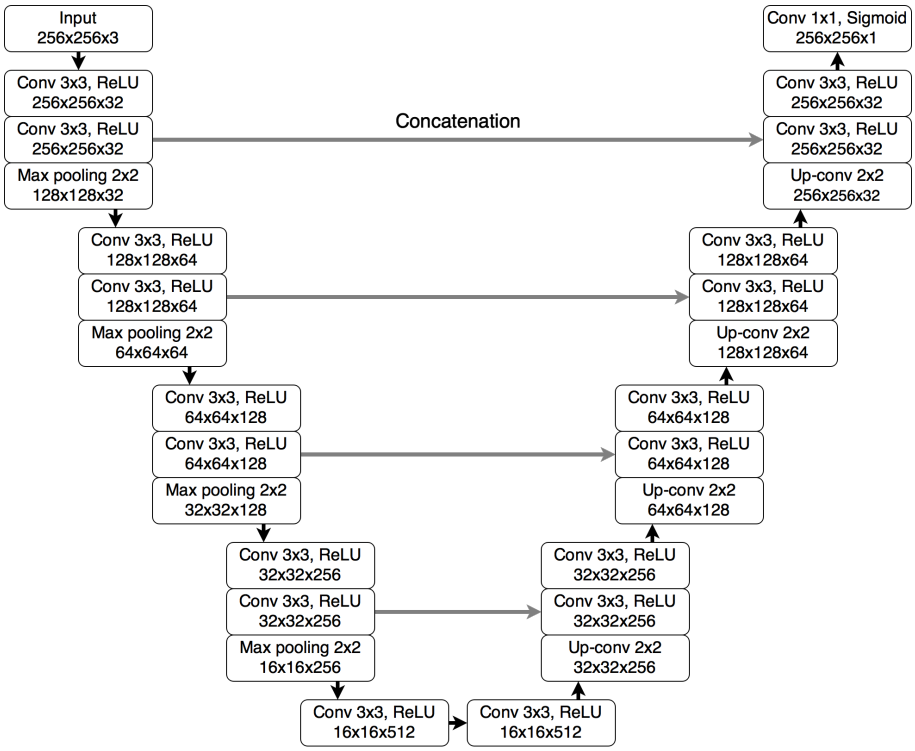


Figure 14: Scheme of the network I have used for experiments.[11]

One of my biggest problems was size of images. The size of the model with all trained weights for such large images was too big for GPU with memory size of 12GB. It was impassible to process whole image in original quality.

For this reason I have decided to try two options how to tackle with this problem. First of them was resize image and lost some data. Second one was to use sliding window method to be able to get maximum utility from original image. Disadvantage of this method was that when we crop part of image we are loosing context, which can be really important. In the figure 15 we can see, how window was sliding through image and that central part was the most exposed one.

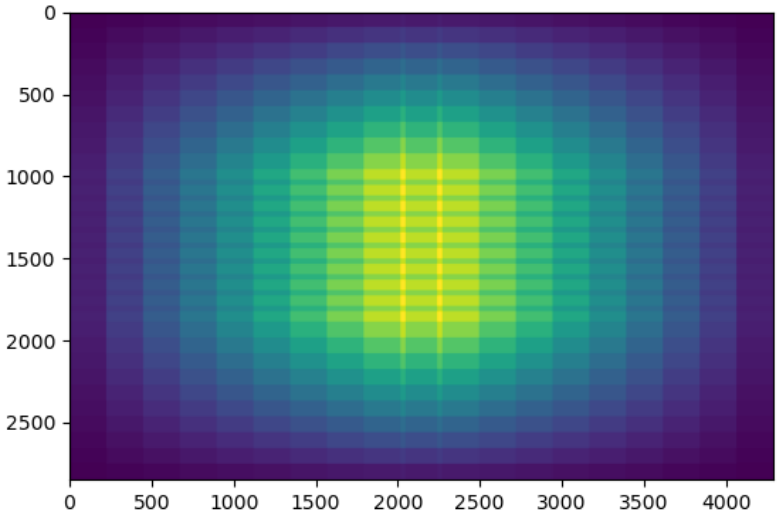


Figure 15: Visualisation of how many times what place in the image is part of the window

For experiments I have used

5.2 Visualisation

To make the results more visible, I decided to visualise the result of each segmentation on an image that has the same dimensions as the original image, using three colours. Areas that should have been segmented but were not marked by the network are marked in blue. Areas that should have been segmented and correctly identified by the network are marked in green. Areas marked by the network, but should not have been marked, are marked in red. As you can see in the picture 16.

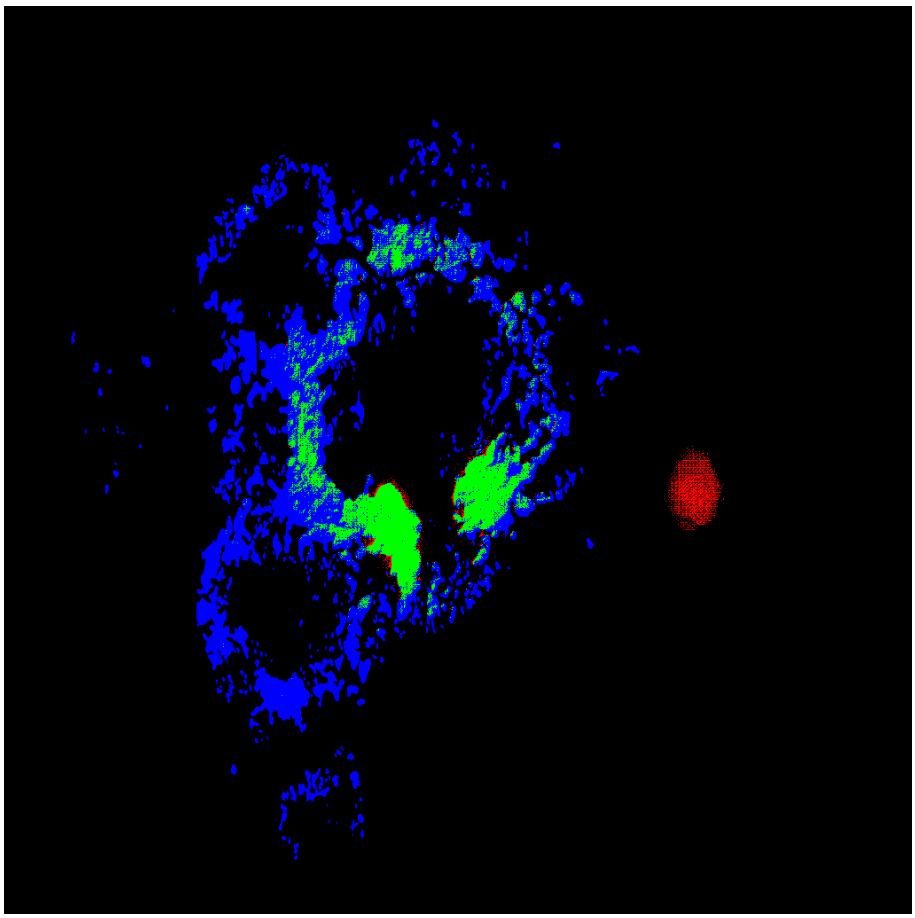


Figure 16: Example of visualisation image segmentation result.

5.3 Calculation of error

In order to calculate, how successful segmentation was I used Jaccard Index also known Jaccard similarity coefficient. [12] This index calculate similarity of two sets. In our case first set is known groundtruth denoted as A and the second set is result of the segmentation denoted as B. Then the formula of the index is:

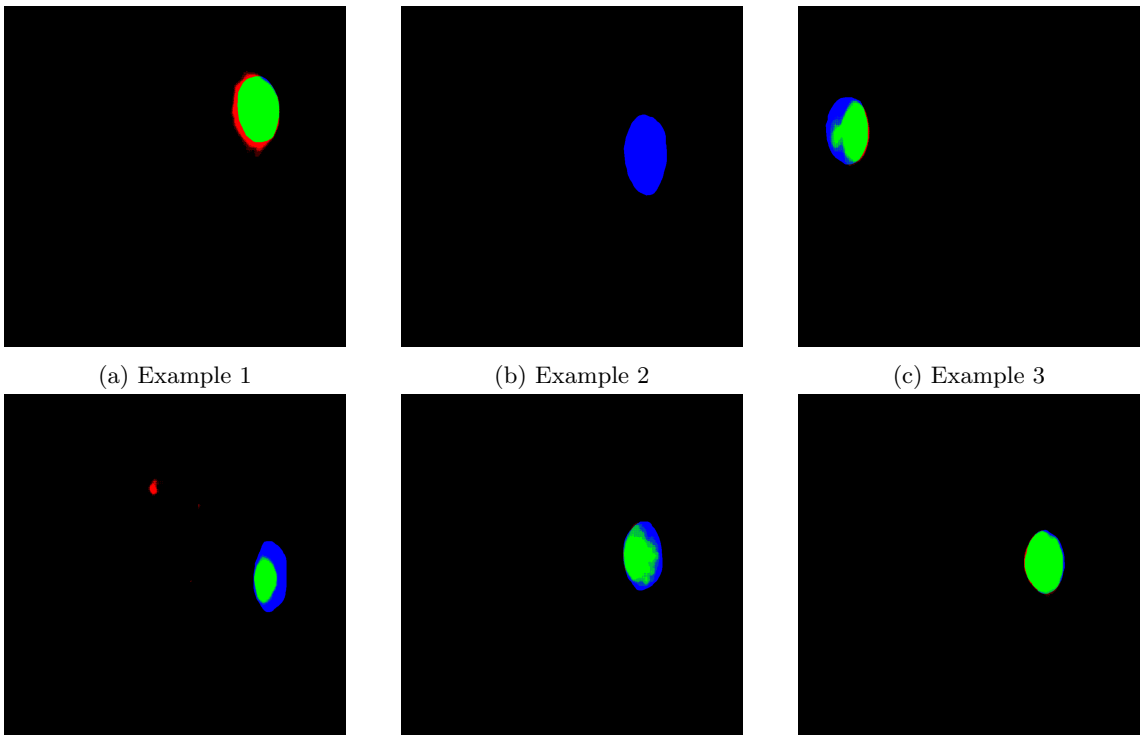
$$J = \frac{A \cap B}{A \cup B} \tag{10}$$

To calculate it first we have to count the number of members which are contained in both sets. Corresponds to green areas in figure 16. This number we have to divide by the number of members in both sets together - corresponds to sum of blue, red and green areas in figure 16.

After that we get number between 0 and 1, where 1 corresponds to best possible result of segmentation - segmentation result is equal to groundtruth, and value 0 means that absolutely nothing was segmented correctly. In order to get result in percentages we can multiply result by 100. Advantage of this method is that its result is independent on image size, so we can compare just how good segmentation is across different image examples.

5.4 Results

First I tried to segment optic disk. It is really distinctive object, so I had assumed that it will be easy for start. For first try I have decided to resize images to 1024 x 1024 px. Results was not so bad in my opinion, but still it can be much better. For example when we look in the figure 17 in the picture (f) we can see, that segmentation went really well and Jaccard Index is almost 1. In the other hand in the picture (b) my network have not found absolutely nothing and Jaccard index is 0.



(a) Example 1 (b) Example 2 (c) Example 3

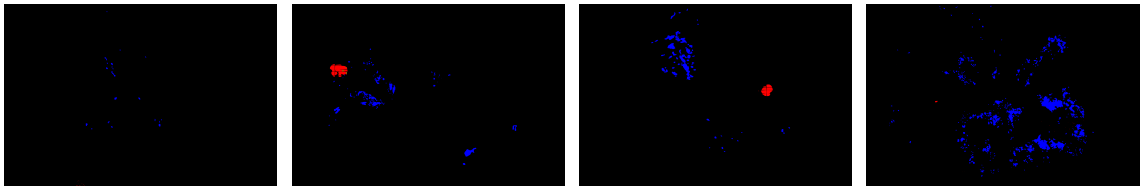
(d) Example 4 (e) Example 5 (f) Example 6

Figure 17: Examples of results of optic disk segmentation.

As a next object I have tried to segment hard exudates. This seems to be more complicated, because by colour they are often quite similar to optic disk. Another problem can be, that their shape is always different.

For this task I have augmented my dataset from 54 to 324 images. I have adjusted brightness, contrast, saturation and hue properties using Collor jitter from PyTorch transformations library. So I added five new version of each original image.

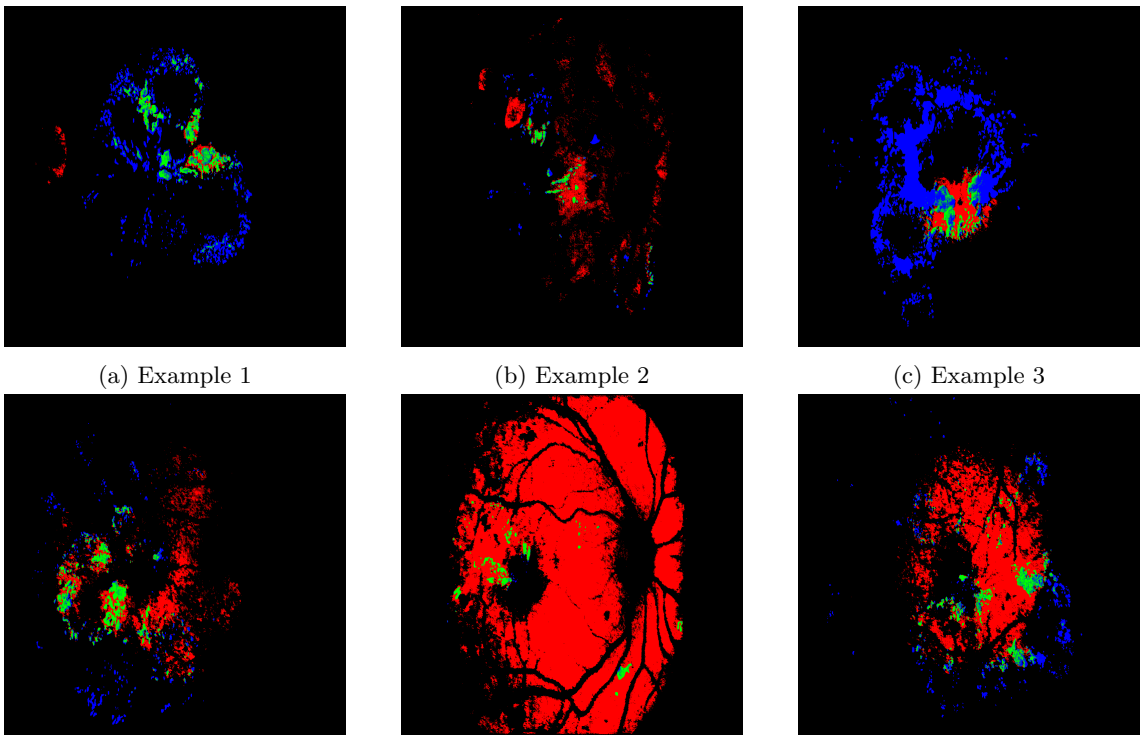
First I tried sliding window method and as you can see in the figure 18 it was not really successful.



(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Figure 18: Examples of results of hard exudates segmentation using sliding window method.

After I have tried segmentation with resized images on size 1024 x 1024 px. Result is better than with sliding window method as you can see in the figure 19. But still Jaccard index score on testing data was around 0.12.



(a) Example 1 (b) Example 2 (c) Example 3

(d) Example 4 (e) Example 5 (f) Example 6

Figure 19: Examples of results of hard exudates segmentation on resized images.

Then I have tried to figure out how to improve learning process. I have seen that loss error is very different for each iteration during one epoch. It is plotted in the chart in the figure 20.

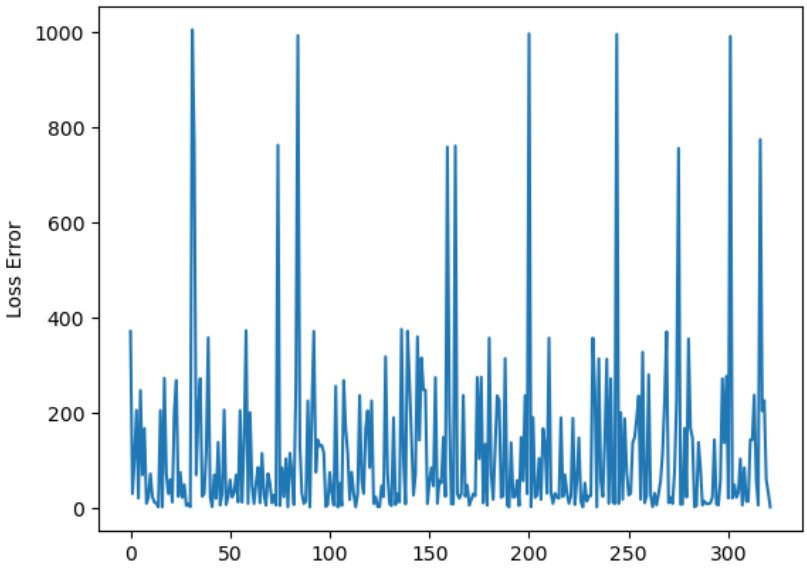


Figure 20: Graph of error loss for each iteration in one epoch.

To be able to observe if there is some progress I have decided to plot following chart 21, where I have plotted only mean of each epoch. I have tried different learning rate parameters and for each variant I ran five epochs.

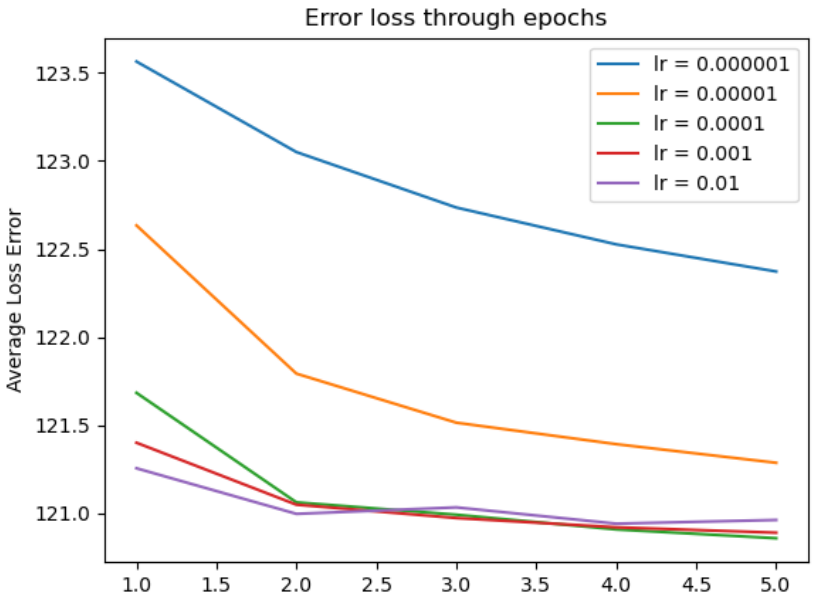


Figure 21: Graph of convergence during learning process for different learning rates.

We can see that the curves converge for a learning rate smaller than 0.01. So for optimal training we would choose a parameter value around 0.00001.

6 Conclusion

To conclude, during this project I have learned many new thing. I got know basics on neural networks especially how convolutional neural networks works. Also now I can better understand how research like this works and which difficulties brings. I am really not satisfied with results of my experiments, I think there is lot to improve. I would like continue in this topic in my bachelor thesis, improve these results and expand this work also for classification part which needs to get understand another branch of neural learning.

References

[1] Alan W. Stitt, Timothy M. Curtis, Mei Chen, Reinhold J. Medina, Gareth J. McKay, Alicia Jenkins, Thomas A. Gardiner, Timothy J. Lyons, Hans-Peter Hammes, Rafael Simó, and Noemi Lois. The progress in understanding and treatment of diabetic retinopathy. *Progress in Retinal and Eye Research*, 51:156–186, 2016. ISSN 1350-9462. doi: https://doi.org/10.1016/j.preteyeres.2015.08.001. URL https://www.sciencedirect.com/science/article/pii/S135094621500066X.

[2] Prasanna Porwal, Samiksha Pachade, Ravi Kamble, Manesh Kokare, Girish Deshmukh, Vivek Sahasrabuddhe, and Fabrice Meriaudeau. Indian diabetic retinopathy image dataset (idrid): A database for diabetic retinopathy screening research. *Data*, 3(3), 2018. ISSN 2306-5729. doi: 10.3390/data3030025. URL https://www.mdpi.com/2306-5729/3/3/25.

[3] Grand Challenge. A platform for end-to-end development of machine learning solutions in biomedical imaging. https://grand-challenge.org/, 2012.

[4] Prasanna Porwal, Samiksha Pachade, Ravi Kamble, Manesh Kokare, Girish Deshmukh, Vivek Sahasrabuddhe, Tom MacGillivray, Désiré Sidibé, Luca Giancardo, Gwenolé Quellec, and Fabrice Meriaudeau. Diabetic retinopathy: Segmentation and grading challenge. https://idrid.grand-challenge.org/, 2017.

[5] Bennett Eye Institute. Bennett eye institute. https://www.bennetteyeinstitute.com/retina-honolulu/diabetic-retinopathy/, 2023.

[6] Vision gallery. Vision gallery. https://www.visiongallerykaty.com/eye-care-services/treating-diabetic-retinopathy/, 2023.

[7] Petr Stádník. My resources on github. https://github.com/PetrStadnik/idrid_segmentation, 2023.

[8] The Linux Foundation. Pytorch transformations. https://pytorch.org/vision/main/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py, 2017.

[9] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

[10] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574, 1959.

[11] Mateusz Buda. U-net for brain mri. https://pytorch.org/hub/mateuszbuda_brain-segmentation-pytorch_unet/, 2023.

[12] Sam Fletcher and Md Zahidul Islam. Comparing sets of patterns with the jaccard index. *Australasian Journal of Information Systems*, 22, Mar. 2018. doi: 10.3127/ajis.v22i0.1538. URL https://journal.acs.org.au/index.php/ajis/article/view/1538.