Assignment 3

Tomas Langebaek Carrizosa. Group members: Richmond Horikawa, Nathan Hu, Chenxiang Zhang, Amalia Riegelhuth

1. Preprocessing, designing network and experiments:

1.1 Describe how you will use the training images to ensure a good generalization to the test images.

Training images are 15 RGB tif images. They have uneven dimensions and don't seem to be a representative dataset for trainig a Convolutional Neural Nework for this particular task. Given this trainig set an image resize is considered, so all images have the same dimensions and they are even in both axis. Because images have 3 channes, only one will be taken onto account after seen which have the most representative data. By doing this, trainig time is expected to improve. Also, by taking only one channel, the traing data will have less infomation that the original data, so the result won't be the as accurate as using all three RGB channels. The same transformation will be applied to the test datset, so a better generalization can be achieved. If generalization is not met with this transformations, a droput could be used in the network and hyper parameters can changes, for example lowered.

1.2 Visualize the images,both in color and all three (RGB) channels separately. Consider if preprocessing might be required to train a good model. If not, describe why not.If yes, describe the preprocessing steps and why you expect them to help.

The images of the three channel are shown bellow:

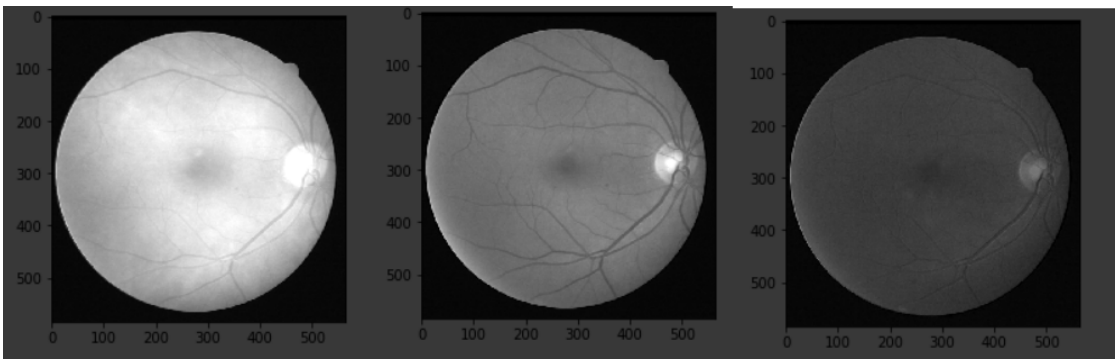

Figure 1: RGB Channels

In this case, taking only the green channel (center) seems like a good idea. This will decrease training time, but not all features will be taken into account. Therefore, not the best result will not be achieved with just one channel. Although this is a problem, the decision to just take the green channel into account was taken. It reduces training time and allows to do more experimentation with the neural network. Images were also resized to be equal size, both training and testing. This can also lead to more generalization. Taking less information into the training model could lead to better result in more general cases for the tests.

1.3 Consider what neural network architecture would be suitable to solve this task and justify your choice. For instance, consider the number, type, and size of layers, what connectivity, what non-linearity, which loss function, which optimizer.

For this particular task, the U-Net network architecture was chosen. The exact graphical model taken from the convolutional neural network lecture is shown bellow.

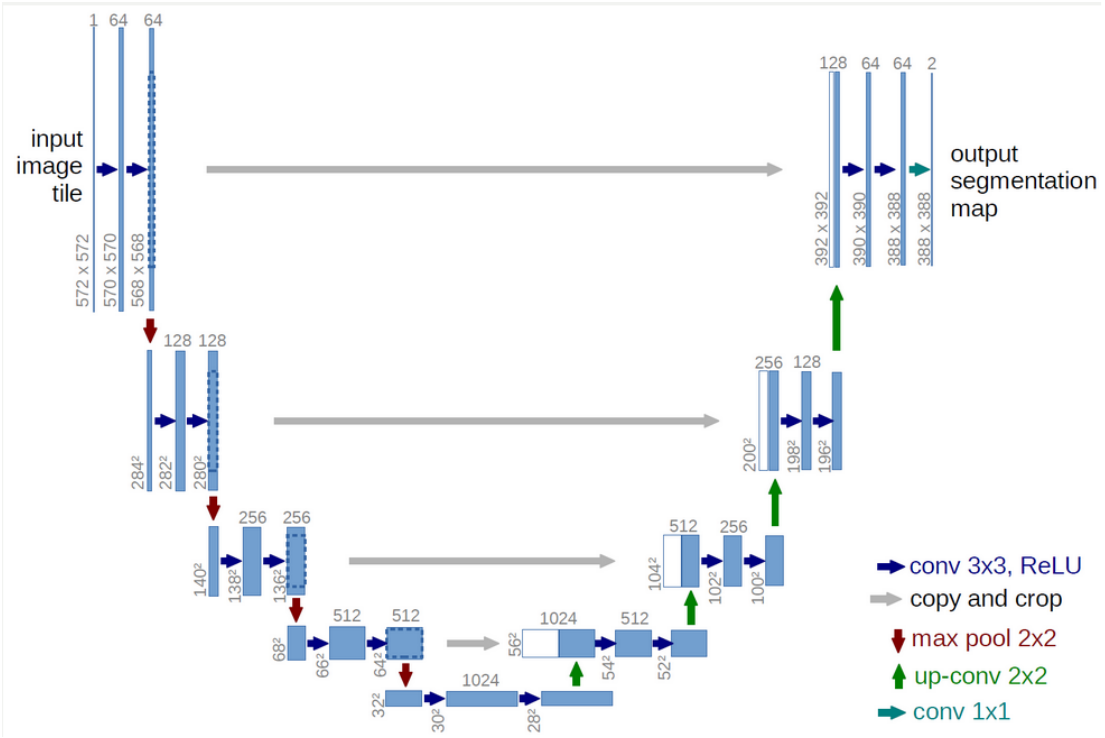

Figure 2: U Net Architecture

As seen in class and in the article "U-Net: Convolutional Networks for Biomedical Image Segmentation" from the university of Freiburg, this a good architecture for medical image processing. An attempt to make an implementation with wider layers was done, but computing processing wasn't enough, so the implementation followed the exact same U Net architecture. It consists in the following: number of layers: 27, type of layers: Convolution, Max Poll and Transposed Convolution layers, size of layers: Minumun size is 64, Maximun

size is 1024 as seen in Figure 2, connectivity: fully connected layers, non-linearity: ReLU, loss function: BCEWithLogitsLoss. Pytorch documentation says the following for this loss function implementation: "This loss combines a Sigmoid layer and the BCELoss in one single class. This version is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.", optimizer: Stochastic gradient descent. The optimizer, Non-linearity and loss function choice was made taking into account the desired output, in this case one channel image. They are also widely used in literature. Inspiration for the Class architechture in python was inspired by https://github.com/milesial/Pytorch-UNet, so code was modular and reusable.

1.4 What is the receptive field of this network? Explain your computation.

The receptive fiel can be calculated using the formula:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

Figure 3: Loss over 9 epoch with 7 training samples

Taken from this blog https://syncedreview.com/2017/05/11/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks/

So for the layers of this particular neural network the receptive field is the following: For the first layer we have (1 inputs + 2*1 padding -3)/2 +1 = 1 For the second is the same, but with 64 inputs

Then we can continue changing the input values and dimension in the max pool operations, so the result is the following for each layer:

$$1 : (1inputs + 2 * 1padding - 3)/2stride + 1 = 1$$

$$2 : (64inputs + 2 * 1padding - 3)/2stride + 1 = 32$$

$$3 : (128inputs + 2 * 1padding - 3)/2stride + 1 = 64$$

$$4 : (128inputs + 2 * 1padding - 3)/2stride + 1 = 64$$

$$5 : (256inputs + 2 * 1padding - 3)/2stride + 1 = 128$$

$$6 : (256inputs + 2 * 1padding - 3)/2stride + 1 = 128$$

$$7 : (512inputs + 2 * 1padding - 3)/2stride + 1 = 256$$

$$8 : (512inputs + 2 * 1padding - 3)/2stride + 1 = 256$$

$$9 : (1024inputs + 2 * 1padding - 3)/2stride + 1 = 512$$

$$10 : (1024inputs + 2 * 1padding - 3)/2stride + 1 = 512$$

UP

$$11 : (1024inputs + 2 * 1padding - 2)/2stride + 1 = 513$$

$$12 : (1024inputs + 2 * 1padding - 3)/2stride + 1 = 512$$

$$13 : (512inputs + 2 * 1padding - 3)/2stride + 1 = 256$$

$$14 : (512inputs + 2 * 1padding - 2)/2stride + 1 = 257$$

$$15 : (512inputs + 2 * 1padding - 3)/2stride + 1 = 256$$

$$16 : (256inputs + 2 * 1padding - 3)/2stride + 1 = 256$$

$$17 : (256inputs + 2 * 1padding - 2)/2stride + 1 = 129$$

$$18 : (128inputs + 2 * 1padding - 3)/2stride + 1 = 64$$

$$19 : (128inputs + 2 * 1padding - 3)/2stride + 1 = 64$$

$$20 : (128inputs + 2 * 1padding - 2)/2stride + 1 = 65$$

$$21 : (64inputs + 2 * 1padding - 3)/2stride + 1 = 1$$

$$22 : (64inputs + 2 * 1padding - 3)/2stride + 1 = 1$$

1.5 Neural networks can be time consuming to train. Consider - and describe - different ways to simplify the problem to keep training time manageable. For instance,do you need to train on all images/whole images/full resolution/all three RGB channels? Feel free to make choices that might lead to less accurate results but would make experimentation insignificantly faster, but make sure to explain your choices and the problems it may cause.

In order to keep the training time manageable, as said before, the original image can be divided into three RGB channels, and the most accurate for the feature we are looking for can be taken. Another approach is to use less than the 15 images into the training process. This will reduce memory usage, and training time will be reduced. The first was chosen for this exercise. For instance, only a batch size of 5 was chosen due to memory getting full with more. Only the green channel was taken as the input in the training. There can be more strategies one can use to keep training time manageable in this case. For example, image resolution can be lowered. This could be good time wise, but it can also lead to higher inaccuracy, more than the two approaches described before. This is due to the fact that vessels get quite small, so a bad resolution doesn't seem like a good idea. A final approach could be to modify the images in different ways; for example, by changing contrast, light and saturation. This takes time to test so these approaches were discarded.

2. Implementation, training, and analysis:

2.1 Implement your net work of choice and train it. Note: you need to make the implementation yourself, downloading and applying a pretrained model is not allowed. Describe your implementation. Use sufficient detail so that someone else can implement your network based on your description.

For this implementation, the following Classes were created:

```
class MaxPool(nn.Module):
  def __init__(self, dimesion):
      super().__init__()
      self.maxPoll = nn.MaxPool2d(dimesion)
  def calculate(self, input):
        result =  self.maxPoll(input)
        return result
```

Following this pattern, these are the other classes, simplified:

```
class DoubleConvolution(nn.Module):

      self.conv = nn.Conv2d(number_inputs, number_outputs, kernel_size=
    kernelSize, padding=1, stride=1)
      self.batch = nn.BatchNorm2d(number_outputs)
      self.relu = nn.ReLU(inplace=True)
      self.conv2 = nn.Conv2d(number_outputs, number_outputs, kernel_size=
    kernelSize, padding=1, stride=1)
      self.batch2 = nn.BatchNorm2d(number_outputs)
      self.relu2 =  nn.ReLU(inplace=True)
  def calculate(self, input):
      result = self.conv(input)
      result = self.batch(result)
      result = self.relu(result)
      result = self.conv2(result)
      result = self.batch2(result)
      result =  self.relu2(result)
```

Another simple convolution class was created, in the same way as the previous example but using just one convolution. The following is the Up Convolution used:

```
class UpConvolution(nn.Module):
      self.upConv = nn.ConvTranspose2d(number_inputs, number_outputs,
    kernel_size=kernelSize, stride=2, output_padding=1)
  def calculate(self, input):
      result =  self.upConv(input)
```

This is the construction of the U Net implemented for this task was done:

```
def __init__(self):
        super(CNN, self).__init__()
        self.conv_1 = DoubleConvolution(1, 64, 3)
        self.maxPool_1 = MaxPool(2)
        ...
```

And the forward function:

```
def forward(self, input):
```

```
        result1 = self.conv_1.calculate(input)
        result2 = self.maxPool_1.calculate(result1)
        ...
```

Finally, for the Crop and Copy step the following function was created:

```
def CopyAndCrop(self, tensor_1, tensor_2):
        extra_dimensions = tensor_2.size()[1] - tensor_1.size()[1]
        tensor = Variable(tensor_2)
        cropped_tensor = tensor[:,0:extra_dimensions,:,:]
        return torch.cat((cropped_tensor, tensor_1), dim=1)
```

2.2 Show how the loss evolves over training epochs. Do the same on a smaller training set. Do you have enough training data to make a good model?

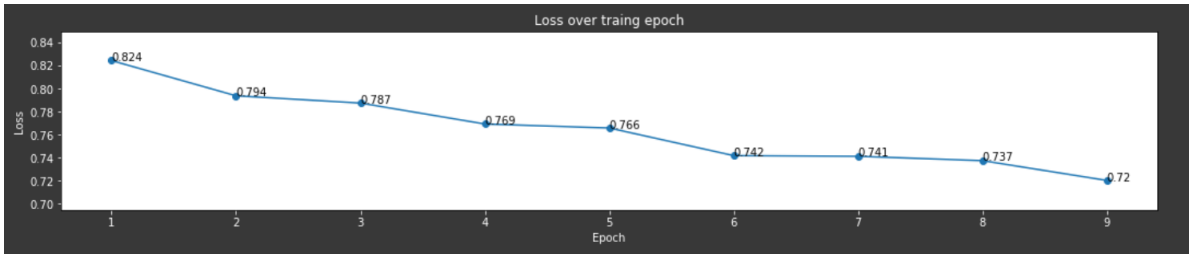

Figure 4: Loss over 9 epoch with 15 training samples

Figure 5: Loss over 9 epoch with 7 training samples

The loss is high in both cases, therefore we can see that increasing the dataset from 7 to 15 doesn't help that much in the decrease of the loss. Therefore multiple epoch are needed if we want to get a good model. So we don't have enough training data to make a good model with a reasonable number of epoch. With more epoch iterations, more time is needed to archive the goal of training a good model.

2.3 Does the model overfit or under fit? Why? Try to make the model overfit by adjusting hyper parameters. Describe your observations.

The model is under fitting when tested with 10 epoch and learning rate of 0.001. Then, seems to work fine with 40 epoch. Another test was done with 5 epoch, both with learning rate of 0.001 and 0.000000001. The result are the following.

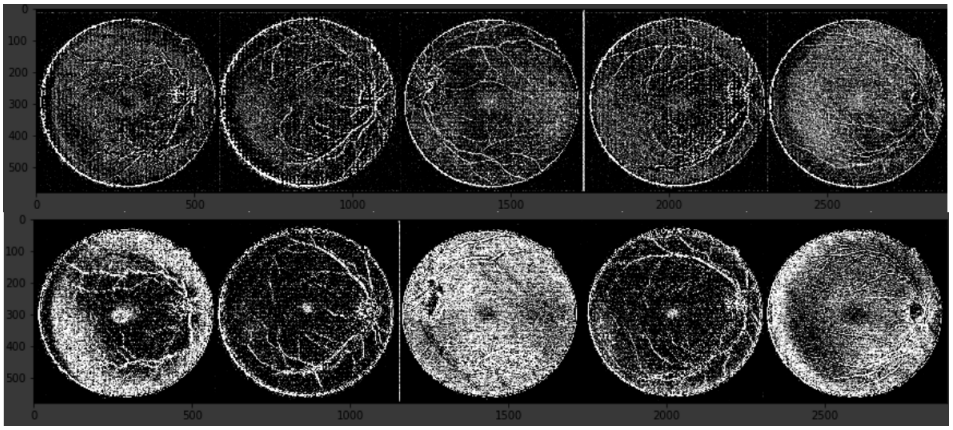

Figure 6: Top: bigger learning rate. Bottom: smaller learning rate

It can be seen that the model tends to not underfit when the learning rate is bigger, therefore it does not overfit with the current learning rate of 0.001 and multiple epoch iterations are applied. On the other had the model is not over fitting because training loss is decreasing as well as testing loss. Adjusting the model hyper parameters so that learning rate is small lead to over fitting results

2.4 Experiment with at least two different loss functions. What do you observe?

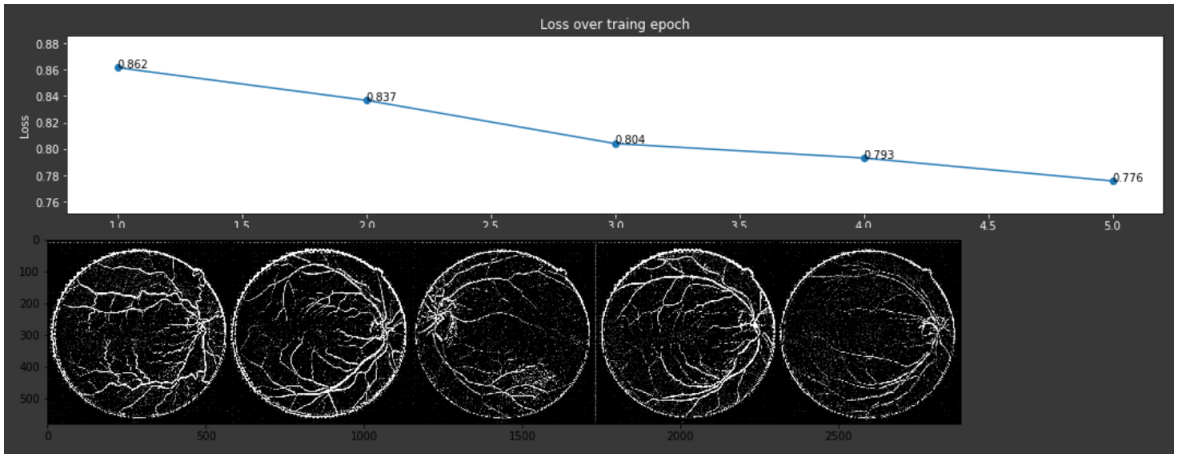

Figure 7: BCEWithLogitsLoss

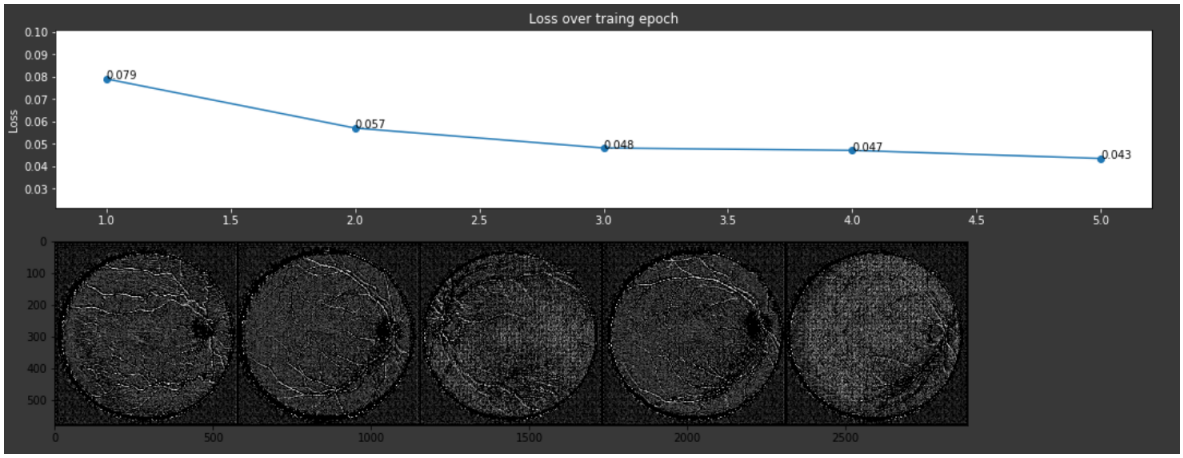

Figure 8: SmoothL1Loss

The first loss function to test was the BCEWithLogitsLoss and the second the SmoothL1Loss. After testing the model with these two function is clear that choosing a loss function is a very import part of the neural network training. In this case, the first function makes a much bigger loss than the second one but drops faster too. This is nos represented in the actual result on the testing sets. So a conclusion that can be made from this results is that it is important for the loss to be decreasing, but the quantity doesn't seem to have as much impact as the drop rate. This can be a generalization, but is the conclusion made from this particular experiment. The second function seems to take more epoch in order to produce a result similar to the one obtained with the first loss function.

2.5 Experiment with at least two sets of choices for the model hyperparameters and try explain the differences you observe.

For this comparison, hyper parameter results are shown in Figures 9, 10 and 11. In figure 9 the learning rate was decreased to 0.000000001 and in figure 11 the momentum was set to 0.3. This choices were taken so that result are comparable with the initial hyper parameters shown in Figure 10.

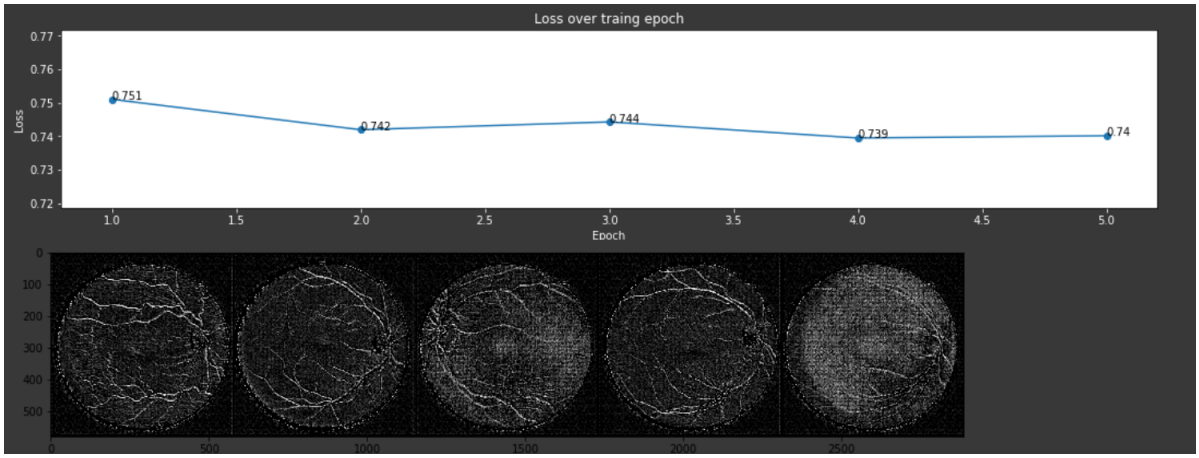

Figure 9: Smaller Learning rate

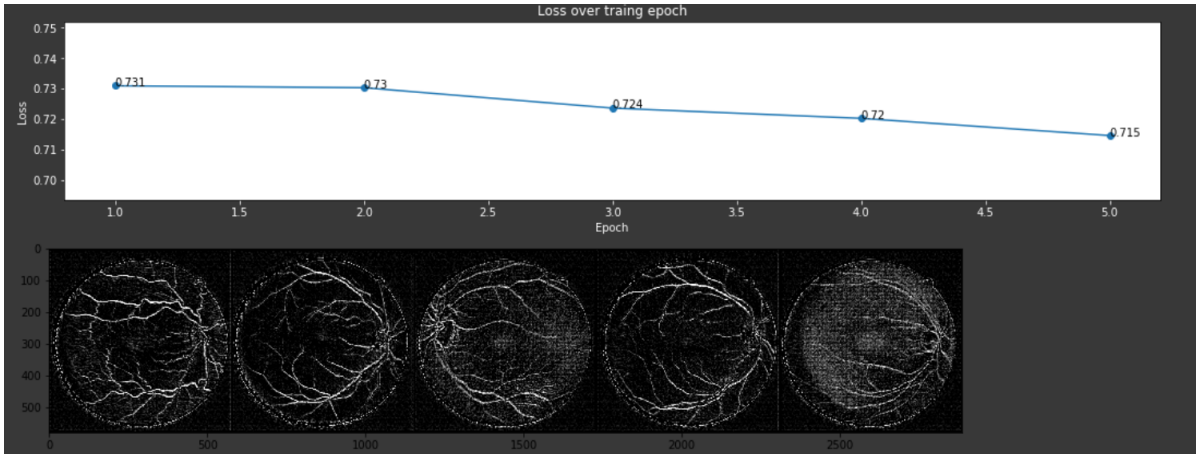

Figure 10: Reference

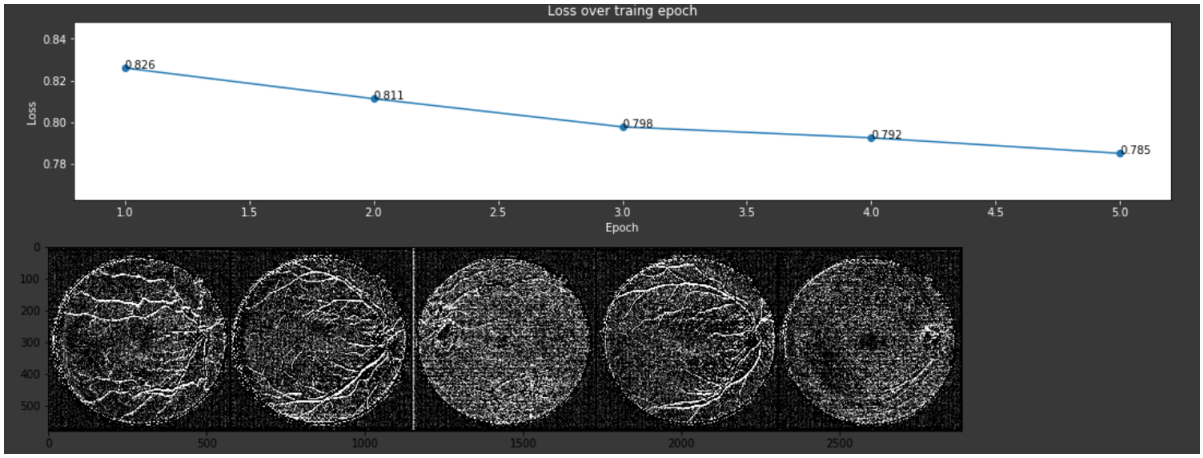

Figure 11: Smaller Momentum

When the learning rate was decreased, the overall loss was higher than the reference result. This is because the learning rate represent the way (or rate) in which the trained model weights change according to the input given, so a smaller learning rate is expected to produce more loss in the beginning. Features are learned slower than the reference. Then, more accuracy is expected in the final result. This also produces less chance of under fitting the model. When Momentum was decreased to 0.3, smaller steps toward the minimum are taken, so there is a higher chance to not finding a good result. Because the momentum is small loss also goes down much faster than the reference. In this case results don't match the prediction of having a higher convergence of the momentum is increased, but this might happen of multiple epoch iterations are done, not just 9. The image for the height momentum also shows good result but lots of noise, so the model might be arriving to a local minimum but still getting to good results.

3. Results evaluation and possible improvements:

3.1 A common performance metric for segmentation is the Dice Similarity Coefficient, defined as the volume of the intersection of the resulting segmentation with the ground truth divided by the sum of the two volumes. Submit your segmented test images for the model that you think performs best. What value of Dice coefficient do you expect this result to have, and why?

This is the final result with test images and epoch = 40 and the valuer for loss:

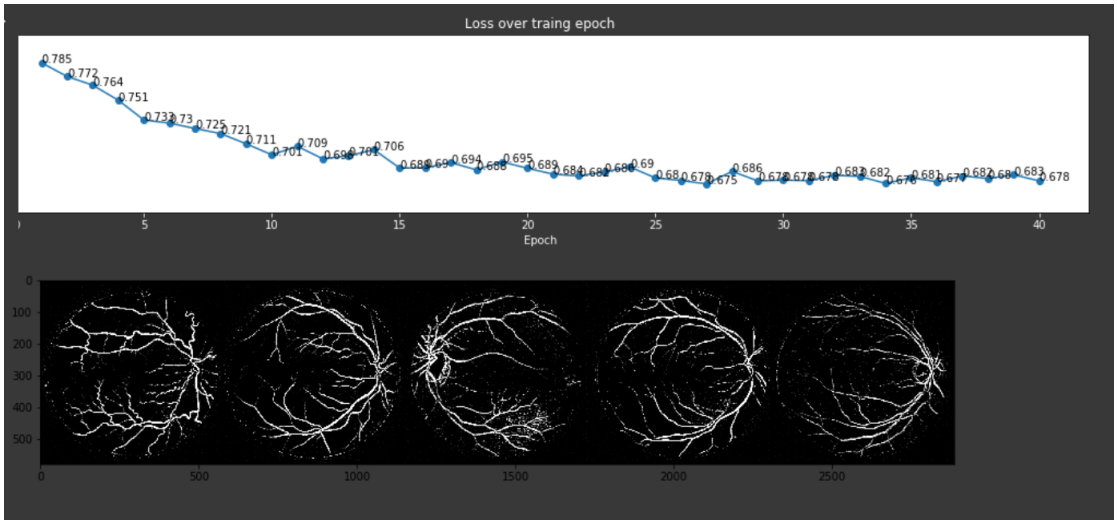

Figure 12: Final result for Test Images

The Dice Similarity coefficient has a minimum value of 0 and a maximum value of 1. Taking into account the images submitted for this assignment, the value of this metric should be around 0.7. This is because the result images have noise, or white dots, in many places where there are no vessels in the space of the retina. Vessels are not as perfect as the Labels for the training set, but the result is still quite good.

3.2 How could you improve these results, if you had more time? List at least two suggestions and justify your answer.

Two approaches could be done if I had more time. First, more pre processing could have lead to better results. As said before, 'filters' and image processing to the training samples could have let to a more accurate model. Image transformations in their light and contrast could have extracted better data from the vessels. This could also improve training time. The second approach has to do with data augmentation. As seen before in this report, I consider the 15 samples as now the amount of data needed to get a good model. Therefore, data augmentation from the training images can improve the results. One last thing that can be done is to do more research to get a better understanding of the way hyper parameters affect the neural network and how each of the different optimizes and loss functions available in Pytorch work, so a better choose can be made in this aspect.