Literature Review

Retinal Vessel Segmentation on DRIVE dataset

Resources

DataSet (Provided by Anna): The Digital Retinal Images for Vessel Extraction (DRIVE) dataset is a dataset for retinal vessel segmentation. It consists of a total of JPEG 40 color fundus images; including 7 abnormal pathology cases. The images were obtained from a diabetic retinopathy screening program in the Netherlands. The set of 40 images was equally divided into 20 images for the training set and 20 images for the testing set. The photographs for the DRIVE database were obtained from a diabetic retinopathy screening program in The Netherlands.

Competition: https://drive.grand-challenge.org/

Download: https://www.kaggle.com/datasets/zionfuo/drive2004?resource=download

Papers:

1. https://arxiv.org/pdf/2105.09365v2.pdf
2. https://ieeexplore.ieee.org/document/9504555
3. https://arxiv.org/pdf/1505.04597v1.pdf

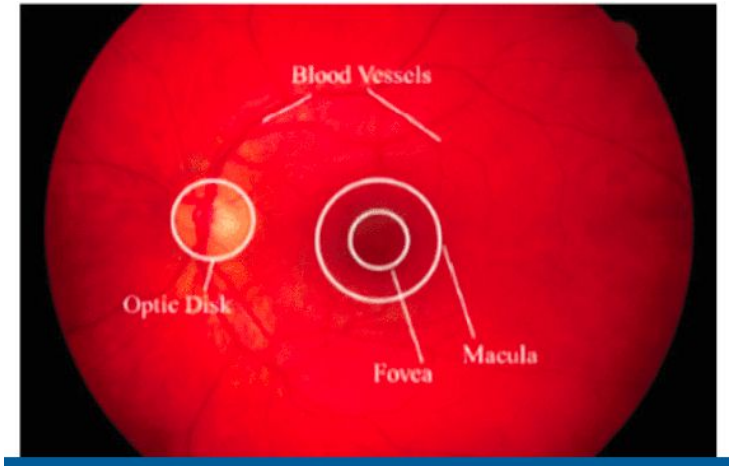
Abstract

Semantic Segmentation is a technique that involves assigning a label to each pixel in an image based on its semantic meaning. For Example, If an image contains cars,buildings,trees and roads the labels typically correspond to a different object classes or regions, such as cars, buildings, trees, and roads. Semantic segmentation can be used to identify and segment various anatomical structures in medical images, including CT scans, MRI scans, X-rays, ultrasound and Retinal images. For example, semantic segmentation can be used to segment organs such as the brain, lungs, liver, or pancreas, to detect and localize tumors or other abnormalities within these organs.

The DRIVE (Digital Retinal Images for Vessel Extraction) dataset is a widely used benchmark dataset in the field of retinal image analysis.Semantic segmentation can be used on the DRIVE dataset to segment the retinal blood vessels and extract important information from the images. The goal of vessel segmentation is to accurately locate the position and shape of the blood vessels in retinal images, which is an important step in the diagnosis of many retinal diseases such as diabetic retinopathy and age-related macular degeneration.

Deep Dive into Data

The retinal vessel map contains abundant geometric characteristics, such as vessel diameter, branch angles, and branch lengths. These geometric characteristics reflect clinical and pathological features, which are used to diagnose hypertension, diabetes, and atherosclerosis.The optics medical practitioner uses retinal blood vessels to diagnose vascular and vascular system lesions related diseases, which interprets diabetic retinopathy (DR) and diabetic maculopathy (MD). These are the leading causes of global blindness.



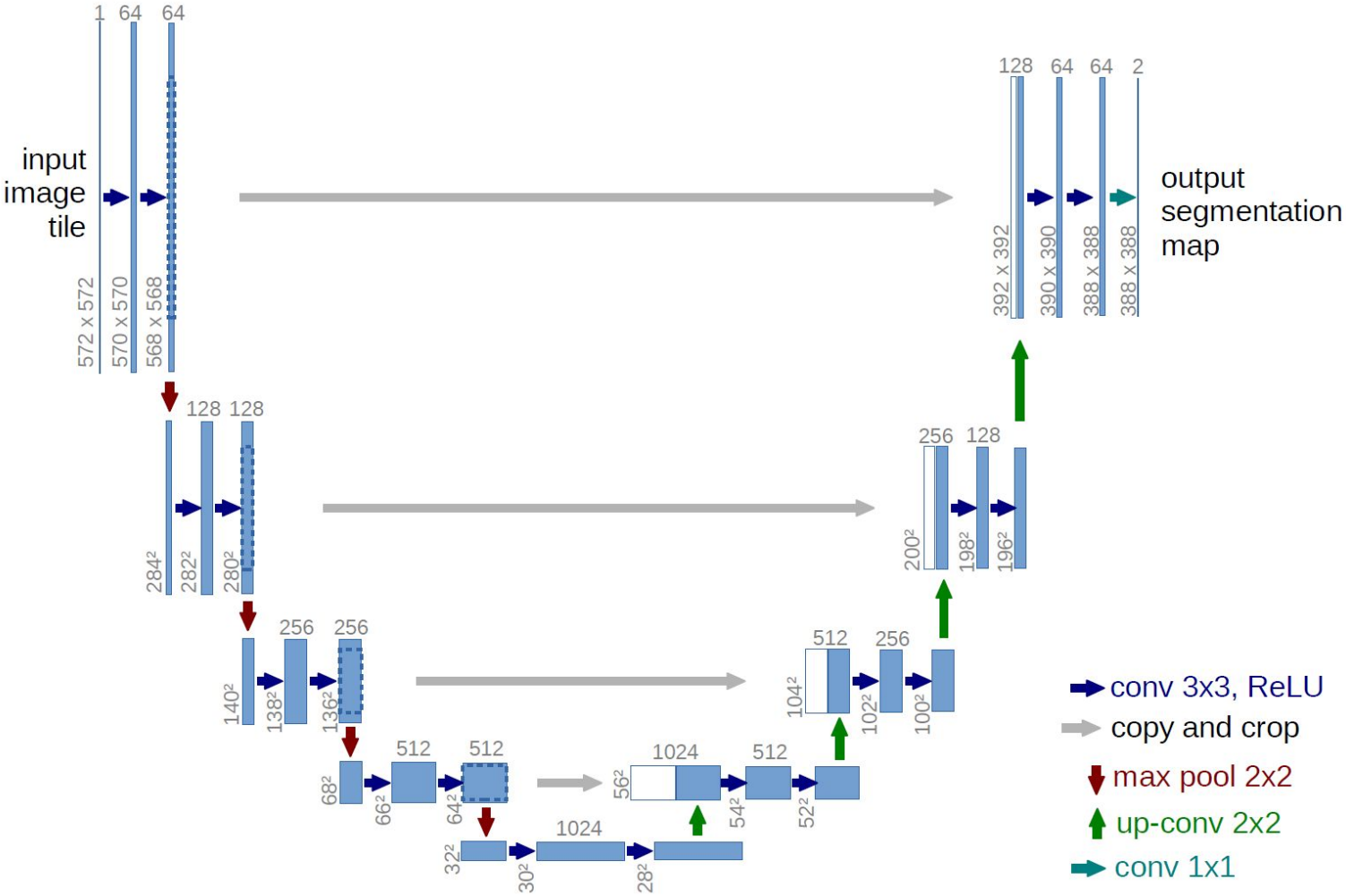
Introduction

The convolutional networks is already existing for a long time , their success is limited due to the size of the available training sets and the size of the considered networks. The typical use of convolutional networks is on classification tasks, where the output to an image is a single class label.However, in many visual tasks, especially in biomedical image processing, the desired output should include localization, i.e., a class label is supposed to be assigned to each pixel.

There is a general perception that successful training of deep networks requires many thousand annotated training samples. Thousands of training images are usually beyond reach in biomedical tasks. I am going to present a network and training strategy that relies on the strong use of data augmentation to use the available annotated samples more efficiently.

The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization.In the below slide you can see the U-net architecture.

U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.


input image tile
output segmentation map
1
64
64
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512
256
104²
102²
100²
256
128
200²
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

The U-net Architecture

U-net architecture are “fully convolutional network”. We modify and extend this architecture such that it works with very few training images and yields more precise segmentations. The main idea is to supplement a usual contracting network by successive layers, where pooling operators are replaced by upsampling operators. Hence, these layers increase the resolution of the output.

In order to localize, high resolution features from the contracting path are combined with the upsampled output. A successive convolution layer can then learn to assemble a more precise output based on this information. One important modification in our architecture is that in the upsampling part we have also a large number of feature channels, which allow the network to propagate context information to higher resolution layers. As a consequence, the expansive path is more or less symmetric to the contracting path, and yields a u-shaped architecture. The network does not have any fully connected layers and only uses the valid part of each convolution, i.e., the segmentation map only contains the pixels, for which the full context is available in the input image.This strategy allows the seamless segmentation of arbitrarily large images by an overlap-tile strategy.

As for our tasks there is very little training data available, we use excessive data augmentation by applying elastic deformations to the available training images. This allows the network to learn invariance to such deformations, without the need to see these transformations in the annotated image corpus. This is particularly important in biomedical segmentation, since deformation used to be the most common variation in tissue and realistic deformations can be simulated efficiently. Another challenge in many cell segmentation tasks is the separation of touching objects of the same class.The resulting network is applicable to various biomedical segmentation problems.

In this slide we will show the result of segmentation on DRIVE dataset. DRIVE dataset consist of Digital Retinal Images for Vessel Extraction.

Data Augmentation

Data augmentation is essential to teach the network the desired invariance and robustness properties, when only few training samples are available. In case of microscopical images we primarily need shift and rotation invariance as well as robustness to deformations and gray value variations. Especially random elastic deformations of the training samples seem to be the key concept to train a segmentation network with very few annotated images. We generate smooth deformations using random displacement vectors on a coarse 3 by 3 grid. The displacements are sampled from a Gaussian distribution with 10 pixels standard deviation. Per-pixel displacements are then computed using bicubic interpolation. Drop-out layers at the end of the contracting path perform further implicit data augmentation.

Dataset

In Retinal Vessel Segmentation problem, the most widely used dataset is DRIVE2 dataset. Each paper in recent years have reported their model performance metrics on this dataset. DRIVE dataset has 20 train and 20 test images. Each image in the training set addresses a different kind of disease. It is quite challenging to create a high-performance model with this amount of annotated data. We use DRIVE dataset to address these problems.

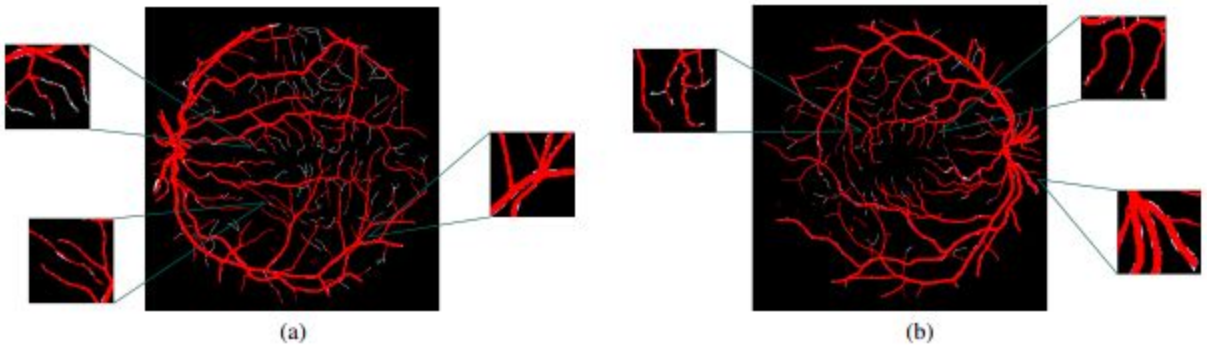
Experimental Setup

Problem Definition:

The problem we are focusing on is the segmentation of retinal images with input data that has quality problems. Moreover, the amount of the annotated data is very limited. Problems about the input data may occur due to various reasons such as illumination, sensor noise, filters of the retinal camera, the input image angle, and other noises. Using the data with such defects limits the capability of the segmentation model. Most of the time the model is unable to segment the regions with noise due to the fact that the model does not have enough information about the regions trying to be segmented. For example, in the DRIVE dataset, each image in the training set addresses a different disease. Each image may come with different quality problems. Combining this with the input image scarcity problem of the medical imaging problems, the problem at the hand becomes even more difficult. If one can identify the problems of the input images well, the quality of the segmentation model can be increased with the help of the data augmentation.

Proposed Methodology

We propose that optimal data augmentation can be successful for Retinal Vessel Segmentation, using simple U-Net architecture . Data augmentation techniques are helpful for three reasons. First, they are helpful because the input data is very scarce. Data augmentation techniques increase the input image size and provide the model some extra information to learn. Second, through data augmentation we can recover some performance loss that occurred in the models due to the image quality. If the practitioner decides the data augmentation technique based on the problems of the input image the segmentation performance can be increased. Third, data augmentation will help with the segmentation model we used, which is the U-Net architecture that makes use of pooling operations. The model learns relatively lower from the corner and side parts of the input image. Data augmentation strategies can address all these three problems. In order to address the third problem, we add rotated versions of the input images to the dataset using various angles. We use data augmentation techniques that rely on adding noise to the original image so that our model can learn more from the noisy images. Noise data come from the normal distribution with mean 0 and standard deviation . In our study we use augmentations with different epsilon values each greater than or equal to 1. Another technique we use is dropout, which targets input pixels. In dropout data augmentation technique, pixels of the input image are set to zero in a random fashion. The portion of the pixel values to be set to zero is a parameter that needs to be defined. It is well-known that minor vessels are one of the hardest regions to segment. Figure 2 shows the predicted and ground truth vessels together. It can be seen that the majority of the incorrect segmentations occur at the minor vessels. An attempt to increase the success of the segmentation model might be to zoom to these regions and add zoomed images to the dataset. Randomly cropping the input image with the random sizes might be another strategy here. We use shifting and flipping of the input image which are also two successful data augmentation techniques. They are widely used with U-Net model training because U-Net uses convolutional filters. Convolutional filters miss the information in the edge of images. Shifting technique pushes the edges of images to the more central part of the image so that the U-Net model can learn the information in the edges of the original image from this augmented image. In order to address the image quality problems that occurred due to the brightness of the input image, we use gamma correction technique



Two examples for a combination of predictions and ground truths. Red pixels are predicted vessels, white pixels are ground truth pixels. By (a), it is observed that the model errors due to the minor vessels. The model performs well in segmenting thick vessels as it is pointed in the right hand side of (a). The model performs better at segmenting the vessels in (b). Minor vessels are segmented better than (a) in this example.

Our Result:

I have executed the network for 50 epoch on GPU. And achieved 97% accuracy.

```
(image_segmentation) abuzar@abuzar-X512JP:~/Harbour.space/Capstone/Image-Segmentation-DRIVE-dataset$ python test.py
  0%|                                                  | 0/20 [00:00<?, ?it/s]
torch.Size([1, 64, 512, 512])
  5%|                                                  | 1/20 [00:06<02:05,  6.58s/it]
torch.Size([1, 64, 512, 512])
 10%|                                                  | 2/20 [00:09<01:16,  4.25s/it]
torch.Size([1, 64, 512, 512])
 15%|                                                  | 3/20 [00:11<00:58,  3.47s/it]
torch.Size([1, 64, 512, 512])
 20%|                                                  | 4/20 [00:14<00:51,  3.23s/it]
torch.Size([1, 64, 512, 512])
 25%|                                                  | 5/20 [00:17<00:44,  2.99s/it]
torch.Size([1, 64, 512, 512])
 30%|                                                  | 6/20 [00:19<00:40,  2.87s/it]
torch.Size([1, 64, 512, 512])
 35%|                                                  | 7/20 [00:22<00:36,  2.81s/it]
torch.Size([1, 64, 512, 512])
 40%|                                                  | 8/20 [00:25<00:32,  2.74s/it]
torch.Size([1, 64, 512, 512])
 45%|                                                  | 9/20 [00:28<00:31,  2.82s/it]
torch.Size([1, 64, 512, 512])
 50%|                                                  | 10/20 [00:30<00:28,  2.80s/it]
torch.Size([1, 64, 512, 512])
 55%|                                                  | 11/20 [00:33<00:25,  2.85s/it]
torch.Size([1, 64, 512, 512])
 60%|                                                  | 12/20 [00:36<00:22,  2.85s/it]
torch.Size([1, 64, 512, 512])
 65%|                                                  | 13/20 [00:39<00:20,  2.91s/it]
torch.Size([1, 64, 512, 512])
 70%|                                                  | 14/20 [00:42<00:16,  2.82s/it]
torch.Size([1, 64, 512, 512])
 75%|                                                  | 15/20 [00:44<00:13,  2.77s/it]
torch.Size([1, 64, 512, 512])
 80%|                                                  | 16/20 [00:47<00:11,  2.77s/it]
torch.Size([1, 64, 512, 512])
 85%|                                                  | 17/20 [00:50<00:08,  2.70s/it]
torch.Size([1, 64, 512, 512])
 90%|                                                  | 18/20 [00:52<00:05,  2.69s/it]
torch.Size([1, 64, 512, 512])
 95%|                                                  | 19/20 [00:55<00:02,  2.68s/it]
torch.Size([1, 64, 512, 512])
100%|                                                  | 20/20 [00:58<00:00,  2.91s/it]
Jaccard: 0.6634 - F1: 0.7974 - Recall: 0.7771 - Precision: 0.8240 - Acc: 0.9657
FPS:  0.39447653064622307
(image_segmentation) abuzar@abuzar-X512JP:~/Harbour.space/Capstone/Image-Segmentation-DRIVE-dataset$
```

Future

Lots of paper have been published till now in the field of Semantic Segmentation.

On the DRIVE dataset, some of the publication claims 97-98% accuracy.
I will try to improve the result by tuning different different deep neural network architecture and by adding more and more data augmentation techniques.