University of CINCINNATI

AUDREN ADRIEN
2023-2024

Deep Learning:
Homework 3

Medical images segmentation: U-net

Content

I. Data

The data set is composed of high-resolution retinal fundus images and for each of them a corresponding pixel-level ground truth annotation. Our goal will be to implement a U-net model that will be trained thanks to these annotations.

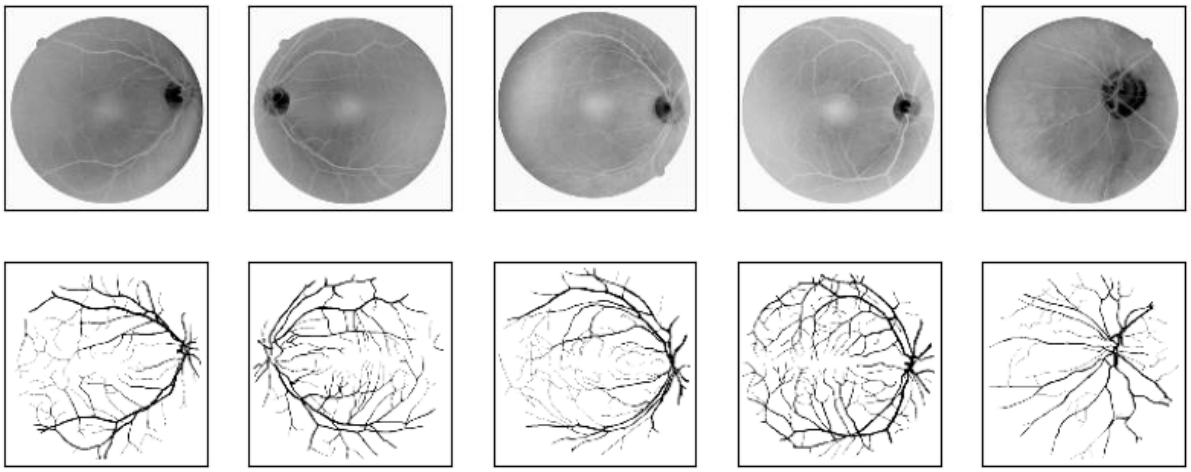


Figure 1.1: 5 normalized images of training set and their corresponding annotations.

1) There is a total of 200 images. 100 real images and 100 annotation images split between a training set (80%) and a test set (20%).
2) The goal of this dataset is to train a segmentation model to recognize retinal vessels.
3) Since each pixel of one channel has a value ranging from 0 to 255, all values are included in these boundaries. Each training data sample is an image of 512x512 pixels ranging from 0 to 255 in three channels. But there is only one channel for annotation images (masks).
4) Each training example is a real medical image. Only the masks can contain errors of annotation. If we consider all images have been annotated correctly, then there is no missing information.
5) The labels are pixel-level annotations. A grayscale ranging from 0 to 255 for each pixel and for each of the 100 images.
6) The training and testing data is already defined by the dataset (80-20). Regarding the validation split, since there are few training images (only 80), the validation ration will be 80-20 too (training: 64, validation: 16, testing: 20).
7) Data is usually centered and reduced before each data analysis or treatment. Here, with grayscale's images, to center the data is not that relevant since pixels values are limited to $[\![0,255]\!]$. We will simply use a common min-max normalization used for images.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

The 512x512 images will be resized into 128x128 images to save GPU resources and allocate them for creating a model with a lot of kernels/parameters. A threshold value will be applied to the masks in order to limit the number of class to 2 (background – vessels).

II. U-net model

A. Model architecture

The exact same architecture of the following image will be used. But using different numbers of layers. We are defining a layer by a horizontal stripe. In case of down sampling, a convolution followed by a max pooling, and in case of up-sampling, a transposed convolution followed by a convolution. The following architecture has four layers.

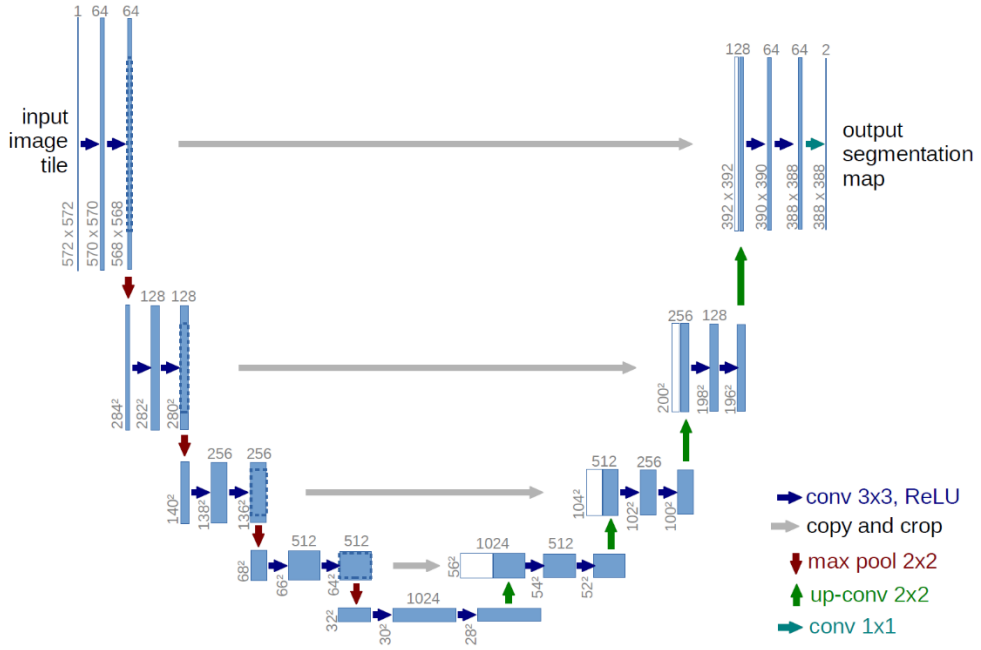


Figure 2.1: 4-layers U-net

The U-net acts first as an encoder which will transform initial input image into a latent representation corresponding to a synthesis of essential information to reconstruct the vessels. And in a second time, acts as a decoder to reconstruct a segmented image. The down sampling is realized by means of max pooling layers and the up sampling by means of transposed convolutional layers.

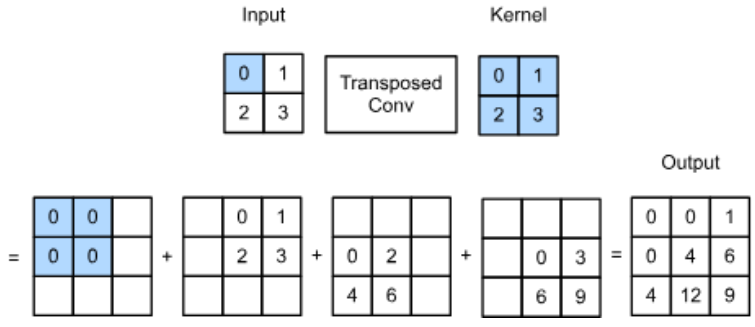


Figure 2.2: transposed convolution explanation

As we go deeper, the network reduces the size of feature maps but produces more and more feature maps with an increasing number of kernels. This is the principle of a convolutional neural network, extracting more and more "useful" features before feeding a fully connected network. But once we reached the desired latent space, we make the size of the features map go up until we reach the initial

size of the input image. Because of the number of layers in this network, we are using residual connections to tackle the gradient vanishing problem.

```
input_layer = tf.keras.layers.Input(shape=(128, 128, 1), name="input_layer")
l1_encode = layers.Conv2D(128, (3, 3), activation='relu', padding="same")(input_layer)
l2_encode = layers.MaxPooling2D((2, 2))(l1_encode)
l3_encode = layers.Conv2D(256, (3, 3), activation='relu', padding="same")(l2_encode)
l4_encode = layers.MaxPooling2D((2, 2))(l3_encode)
l5_encode = layers.Conv2D(256, (3, 3), activation='relu', padding="same")(l4_encode)
l6_encode = layers.MaxPooling2D((2, 2))(l5_encode)
l7_encode = layers.Conv2D(512, (3, 3), activation='relu', padding="same")(l4_encode)
l8_encode = layers.MaxPooling2D((2, 2))(l7_encode)
l9_encode = layers.Conv2D(1024, (3, 3), activation='relu', padding="same")(l8_encode)
l8_decode = layers.Conv2DTranspose(1024, (2, 2), strides=(2, 2), activation='relu')(l9_encode)
l7_decode = layers.Conv2D(512, (3, 3), activation='relu', padding="same")(l8_decode)
l7_decode = layers.Concatenate(axis=-1)([l7_decode, l7_encode]) # skip connection
l6_decode = layers.Conv2DTranspose(512, (2, 2), strides=(2, 2), activation='relu')(l7_decode)
l5_decode = layers.Conv2D(256, (3, 3), activation='relu', padding="same")(l6_decode)
l5_decode = layers.Concatenate(axis=-1)([l5_decode, l5_encode]) # skip connection
l4_decode = layers.Conv2DTranspose(512, (2, 2), strides=(2, 2), activation='relu')(l7_decode)
l3_decode = layers.Conv2D(256, (3, 3), activation='relu', padding="same")(l4_decode)
l3_decode = layers.Concatenate(axis=-1)([l3_decode, l3_encode]) # skip connection
l2_decode = layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), activation='relu')(l3_decode)
l1_decode = layers.Conv2D(128, (3, 3), activation='relu', padding="same")(l2_decode)
l1_decode = layers.Concatenate(axis=-1)([l1_decode, l1_encode]) # skip connection
output_layer = layers.Conv2D(1, (1, 1), activation='sigmoid', padding="same")(l1_decode)
```

Script 2.1: Tensorflow implementation of a 4-layers U-net

We can notice the use of a sigmoid activation function for the output layer. The reason for it is that we want to obtain for each pixel a belonging probability to either background or vessels.

B. Model evaluation

In order to assess our model, we are going to use two metrics well adapted for comparing a model's performance based on two sets of elements A and B.

Dice coefficient: $D = 2\frac{|A \cap B|}{|A|+|B|}$

In our case, sets A and B can be seen as a true mask and a predicted mask. They each contain pixels either completely black or completely white. To compute the intersection, we only need to apply an element-wise multiplication between the two images and sum all the ones.

IoU score: $J = \frac{|A \cap B|}{|A \cup B|}$

(Intersection over union), also called Jaccard coefficient. To compute union, we are simply using: $A \cup B = A + B - A \cap B$.

Model	Dice	IoU
U-net 2 layers	0.28	0.16
U-net 3 layers	0.67	0.5
U-net 4 layers	0.28	0.16

All the performances are assessed on test data.

III. Objective

The problem we are dealing with is a binary classification problem. For each image there are 512x512 pixels to classify either in class "vessels" or "background". If we note n the number of examples available for the training, then, there are $512 \times 512 \times n$ examples (x_i, y_i) with x_i the feature of example i and y_i its label.

We are searching for the best parameters to maximize the likelihood of our classification model.

$$\underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} P(Y = y_i|x_i, \theta)^{y_i}(1 - P(Y = y_i|x_i, \theta))^{1-y_i}$$

Since the logarithm is an increasing function, the problem is equivalent to:

$$\underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} y_i \log\big(P(Y = y_i|x_i, \theta)\big) + (1 - y_i)\log(1 - P(Y = y_i|x_i, \theta))$$

$$= \underset{\theta}{\operatorname{argmin}} \quad -\sum_{i=1}^{n} y_i \log\big(P(Y = y_i|x_i, \theta)\big) + (1 - y_i)\log(1 - P(Y = y_i|x_i, \theta)) \qquad (1)$$

This is called the log-likelihood. This metric allows us to deal with binary classification problems using a logistic regression. In fact, in machine learning the probability $P(Y = y_i|x_i, \theta)$ is obtained using the sigmoid function:

$\sigma(a) = \frac{1}{1+e^{-a}}$, where a is the output returned by our model. This function then gives us a belonging probability. It belongs to class 1 with probability $P(Y = 1|x_i, \theta) = \sigma(a = f(x_i))$.

IV. Optimization

A default optimizer on Tensorflow. Adam is efficient enough to lead optimization of our model's parameters.

V. Model Selection

To avoid overfitting, we can plot validation curve to set an early stopping.

Again, all the performances are assessed on test data.

Model	With normalization		Without normalization	
	Dice	IoU	Dice	IoU
U-net 2 layers	0.28	0.16	0.0	0.0
U-net 3 layers	0.67	0.5	0.28	0.17
U-net 4 layers	0.28	0.16	0.0	0.0

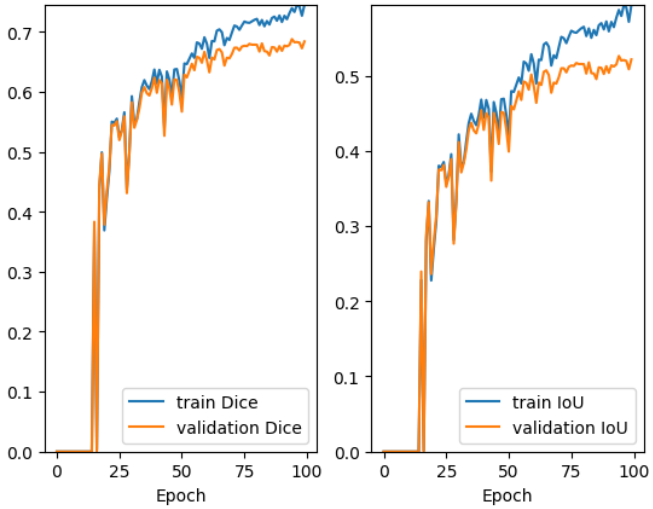


Figure 5.1: 3-layers U-net performances on 100 epochs

Results clearly show us the importance of scaling data before training.

The validation curve remains stable after 75 epochs. It is not necessary to continue the training of the model.

VI. Model Performance

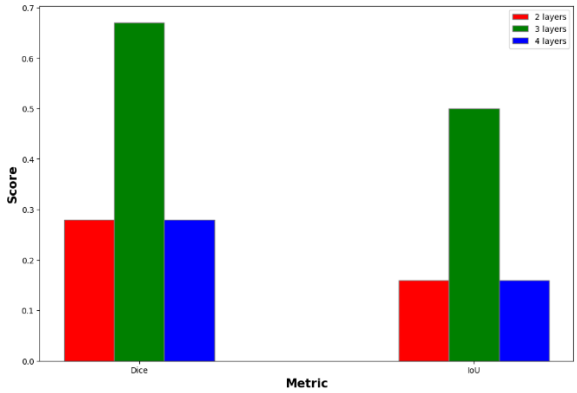
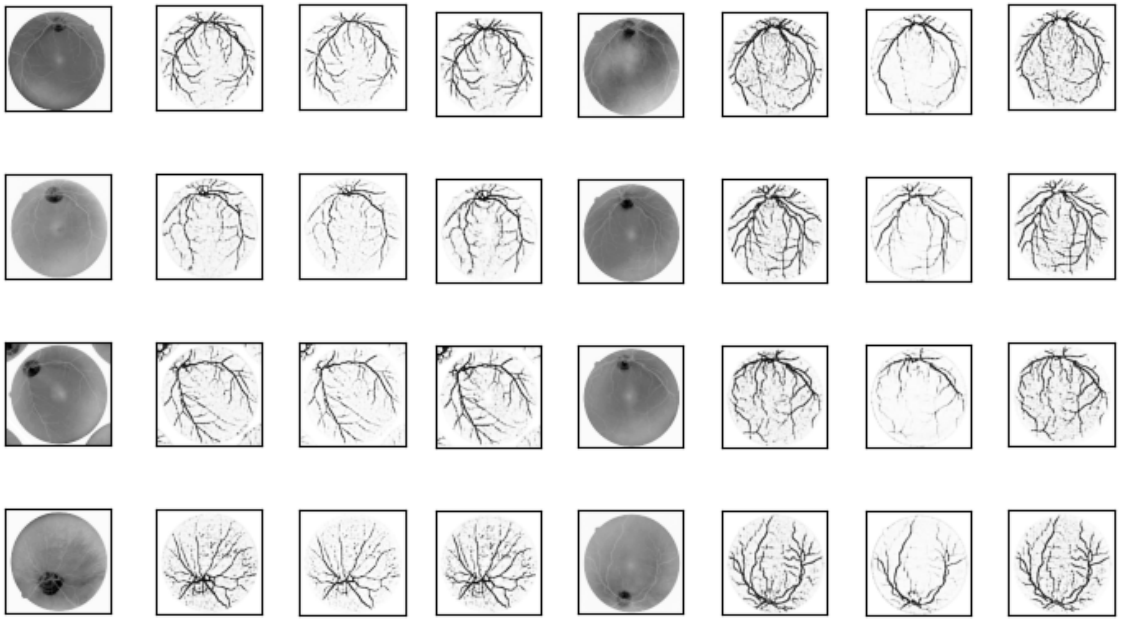


Figure 6.1: Bar plot of test performances for different models



Train predictions: 2L, 3L, 4L

Test predictions: 2L, 3L, 4L

Conclusion

Each image has been resized into a 128x128 pixels instead of initially 512x512. This enables us to save many GPU resources to be able to train a model with many kernels and layers, the U-net. As shown in section 5, scaling the data before any training is extremely important to obtain good results. The model with three layers seems to have the best performance (Dice and IoU). During training, I noticed the way the model was learning was very random. To avoid obtaining too many different results I set the seed of Tensorflow to a fixed value. Predictions of section 6 show us that even though images are very similar to real annotations, the dice and IoU scores remain low because some pixels keep a low but non null intensity and are therefore classified as “vessels”. The model with 3 layers has the best performances because the gray disk almost disappeared from predictions.