U-Net: Convolutional Network for Image Segmentation

12/04/2021

Bruna Silveira Goncalves

**ABSTRACT**

This report discusses convolutional neural networks, which have the ability to label to each pixel of an image. This was created by Olaf Ronneberger, Philipp Fischer, and Thomas Brox at University of Freiburg. This was a breakthrough in image segmentation, as it gives a label for each pixel through a extend convolutional network which is known as U-Net. The implementation is very important for the biomedical field. This report applied the network to cells in order to find the precise edge of a cell. Hence, the cell can be easily measured, and, according to how it stretches, one can determine if the cell is cancerous or not.

**INTRODUCTION**

In the biomedical field, the visualization and acquisition of images is a challenge. Many times, the pictures obtained are not high resolution and so it is difficult to determine the separation between objects. Another factor is the limited quantity of the data available, due to a variety of reasons including cost, difficulty in obtaining samples from patients, and rarity of some diseases.

Chart

Description automatically generatedConvolutional networks are a perfect network for classifying images. They are very good at picking up on patterns in the input image, such as lines, gradient, circles, or even eyes and faces. Typical convolution networks label the entire picture by designating whether the image contains a particular object or not. However, the strategy used in this U-Net application was to generate more precise segmentations by labeling each pixel. The main idea of this application was to decrease the number of pixels while at the same time increasing the number of filters, until the image is 32x32 pixels and has had 1024 filters applied. Next, the model starts to return to the number of pixels and filters of the original image, in this case 512x512 and 1 filter. The shape of the model is similar to the letter ‘U’ as you can see in Figure 1, hence the name U-Net.

FIGURE 1: ARCHITECTURE OF U-NET MODEL

The downsampling block consist of 3x3 convolutions, 2x2 max pooling with stride 2, and droupout of 0.2. At each downsampling step, the number of feature channels in the convolution layers double. When the image gets to 32x32 pixels and 1024 filters, the model starts the process to return the original shape. The upsampling block takes the prior image and applies a 2x2 convolution (“up-convolution”) that splits the number of feature channels, followed by a concatenation with the correspondingly cropped image from the contracting path, and two 3x3 convolutions, each followed by a rectified linear unit (ReLU) [1]. At the final layer, a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers and a total of 31,030,593 neurons. This implementation is different from the one presented in the paper by Chute et al. (2010) due to the addition of droupout in the downsampling blocks as well as restoring the output images back to the original size of the input images.

Logo

Description automatically generated with medium confidenceThe data was obtained through Dr. Deo’s research on how cancer cells stretch under certain conditions. However, the images are low quality and blurry, so it became a challenge to measure the size of the diameter of the cells. Shown in Figure 2 is a row of images including the original data, a mask of the image, and the overlay. The model’s goal is to process pictures and highlight the cell and tubes, consequently making it easier to measure.

FIGURE 2: ROW DATA WILL BE TRAINING

Another important aspect of this operation was the employment of data augmentation, implemented by applying elastic deformations to the 51 available training images to create additional images. This is particularly important in biomedicine both because of the aforementioned scarcity of available pictures as well as the resulting accuracy, since it is able to efficiently simulate deformation, which is a common variation in tissue.

**METHODS**

The first step of the implementation was decreasing the images’ data. My original image was 1028x922 and had to be cropped to 512x512. As long as the cell was within the cropped image, this does not affect the process. Another important step was increasing the contrast which gave a better resulting image.

To create the mask image, I used the image segmentation interface of MATLAB. The difficulty of creating mask images by hand bottlenecked the number of images available for processing, which is why data augmentation was so important. The data augmentation was implemented with rotation of range of 15, horizontal and vertical flip, zoom range of 0.2 and the width and height shift range of 0.05.

The model was trained using 100 epochs each containing 10 images, generating a total of 1000 images. Increasing the number of images in each epoch did not lead to better results, likely due to the model overfitting the images because of the data augmentation.

The model did not run well at the beginning of the training, as early stopping occurred due to the variation of loss, so I decided to proceed without using early stopping. Another step was the decision to use binary cross entropy loss due to the mask consisting of only black or white color. The optimization was selected according the U-Net paper, with stochastic gradient descent as high momentum of 0.99 and learning rate of 0.01.

After reading a few papers, I found that most used IoU (Intersection over Union) for their metrics application. It is essentially a method to quantify the percent overlap between the target mask and the predication output. Or in simpler terms, the IoU metric measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across both masks [2]. The ideal value for IoU is 0.

I followed the U-Net paper in selecting a batch size of 1, data set size of 30, and using my test data as validation data also.

A picture containing logo

Description automatically generated**RESULTS AND DISCUSSION**

FIGURE 3: TRAINING RESULTS WITH 30 IMAGES

A picture containing graphical user interface

Description automatically generatedFIGURE 4: TRAINING RESULTS WITH 51 IMAGES

**A picture containing text, map

Description automatically generated**FIGURE 5: RESULTS USING THE SAME DATA AS U-NET

It is clear that more data is needed for the model to perform as expected. Analyzing the images above, the model with 51 images clearly outperforms the model with 30 images. The model with 51 images started to identify other anomalies around the cell as well. Another proof that there was not enough data was that the model’s prediction trained by the same images as the U-Net model did well, as shown in Figure 5.

The convolutional neural network was able to be trained with only 30 images from U-Net while even 51 of the cancer cells resulted in poorer performance. This could be due to the U-Net images having much denser information than the cancer cell images which have only one cell, resulting in more labels per image.

**CONCLUSION**

This project produced a model similar to one presented in U-Net: Convolutional Networks for Biomedical Image Segmentation by Olaf Ronneberger, Philipp Fischer, and Thomas Brox. The images utilized were provided by Dr. Deo’s research at Norfolk State University about how cells stretch under specific conditions. However, the pictures acquired did not have high enough quality to be able determine the location of the edge exactly. Therefore, it requires image segmentation just as many biomedical images do.

In this application, it was clear that more data was necessary, based on the difference between the results of 30 images versus 51 and how it performed when using the same images as the one used by the U-Net paper.

**REFERENCES**

[1] Chute, C., Zheng, J., Ogren, P., Masanz, J., Savova, G., Sohn, S., & Kipper-Schuler, K. (2010). *Mayo clinical Text Analysis and Knowledge Extraction System (cTAKES): architecture, component evaluation and applications* (pp. 507-513). Retrieved from <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2995668/>

[2] JORDAN, J. (2018). *Evaluating image segmentation models.*. Jeremy Jordan. Retrieved 6 December 2021, from https://www.jeremyjordan.me/evaluating-image-segmentation-models/.

**APPENDIX**

CODE

! pip install git+https://github.com/karolzak/keras-unet

from keras\_unet.models import vanilla\_unet,custom\_unet

from keras\_unet.metrics import iou, iou\_thresholded

from keras\_unet.utils import plot\_imgs

import matplotlib.pyplot as plt

import tensorflow.keras.utils as conv\_utils

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import EarlyStopping, ModelCheckpoint

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

from keras.layers import Activation, Dropout, Flatten, Dense

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.optimizers import Adam, SGD, RMSprop

from keras\_unet.metrics import iou, iou\_thresholded

import tensorflow as tf

import os

import cv2

from PIL import Image

import numpy as np

from numpy import asarray

import pandas as pd

from google.colab import drive

drive.mount('/content/drive')

# IMAGE CUT

# left = 350

# right = 862

# top = 200

# bottom = 712

# c = 0

# image\_directory = '/content/drive/MyDrive/Cancer\_Dectetin/f/'

# crop = os.listdir(image\_directory)

# for a in crop:

# #Remember enumerate method adds a counter and returns the enumerate object

#   image = Image.open(image\_directory + a)

#   image = image.crop((left,top,right,bottom))

#   a = image.save(r'/content/drive/MyDrive/Cancer\_Dectetin/input/train/IMAGE/Done' + str(c) + '.jpg')

#   c = c + 1

dataset  = np.zeros([512,512])

label  = np.zeros([512,512])

image\_directory = '/content/drive/MyDrive/Cancer\_Dectetin/INPUT\_OFICIAL/TRAIN/'

Error = os.listdir(image\_directory + 'IMAGE\_TRAIN\_A')

for a in Error:    #Remember enumerate method adds a counter and returns the enumerate object

    image = Image.open(image\_directory + 'IMAGE\_TRAIN\_A/'+ a)

    # image = image.crop((left,top,right,bottom))

    # width, height = image.size

    image = image.convert('L')

    image = asarray(image)

    image = image/255

    dataset = np.append(dataset,image,axis = 0)

    image = Image.open(image\_directory + 'LABEL\_TRAIN\_A/' + a)

    # image = image.crop((left,top,right,bottom))

    # width, height = image.size

    image = image.convert('L')

    image = asarray(image)

    image = image/255

    # image = image.reshape(512, 512,1)

    label = np.append(label,image,axis = 0)

print(len(dataset))

print(len(label))

print(label.shape)

dataset = dataset.reshape((52,512,512))

label = label.reshape((52,512,512))

print(dataset.shape)

print(label.shape)

dataset = np.delete(dataset,0,0)

label = np.delete(label,0,0)

print(dataset.shape)

print(label.shape)

plot\_imgs(org\_imgs = dataset, mask\_imgs = label, nm\_img\_to\_plot = 10, figsize = 6)

ds = dataset.reshape((51,512,512,1))

lb = label.reshape((51,512,512,1))

print(ds.shape)

print(lb.shape)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(ds, lb, test\_size = 0.3)

model\_1 = custom\_unet(input\_shape=(512, 512, 1),use\_batch\_norm = False,

                    num\_classes=1,filters=64,dropout = 0.2,

                    output\_activation = 'sigmoid')

model\_1.summary()

model\_1.compile(loss='binary\_crossentropy',

              optimizer=SGD(lr=0.01, momentum = 0.99),

              metrics=[iou,iou\_thresholded])

from keras\_unet.utils import get\_augmented

train\_generator = get\_augmented(x\_train,y\_train, batch\_size = 2,

                               data\_gen\_args = dict(

                               rotation\_range=15,width\_shift\_range=0.05,

                               height\_shift\_range=0.05,

                               shear\_range=50,

                               zoom\_range=0.2,

                               horizontal\_flip=True,

                               vertical\_flip=True,

                               fill\_mode='constant'))

model\_1.fit\_generator(train\_generator,

                    steps\_per\_epoch=10,

                    epochs=100,

                    validation\_data = (x\_test,y\_test),

                    # callbacks=[es\_monitor]

                    )

predicted = model\_1.predict(x\_test)

print(predicted.shape)

scores = model\_1.evaluate(x\_test, y\_test, verbose=0)

print(scores)

predicted = predicted.reshape((x\_test.shape[0],512,512))

y\_test = y\_test.reshape((x\_test.shape[0],512,512))

print(predicted.shape)

print(y\_test.shape)

plot\_imgs(org\_imgs = x\_test, mask\_imgs = y\_test, pred\_imgs = predicted, nm\_img\_to\_plot = 5, figsize = 9)