# Semantic segmentation with Deep Learning

U Net for Image Segmentation

---

**Chaitanya Viriyala**
Option : DATASIM
École Centrale de Nantes (France)

---

## Abstract

For quantitative analysis, cells in microscopic images need to be segmented from the image. In this notebook we will model the problem in terms of semantic segmentation and approach it by means of deep learning, and more specifically with the U-Net architecture.

# Outline

# 1 Introduction

The goal of semantic segmentation is to label each pixel of an image with a corresponding class of what is being represented.

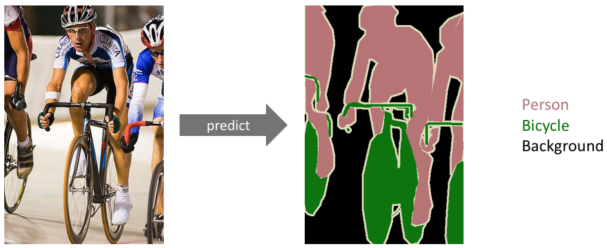


Figure 1: Semantic Segmentation

So, we are particularly interested in biomedical image segmentation, and U-Net, which is a convolution neural network is fast and precise for segmentation of images.

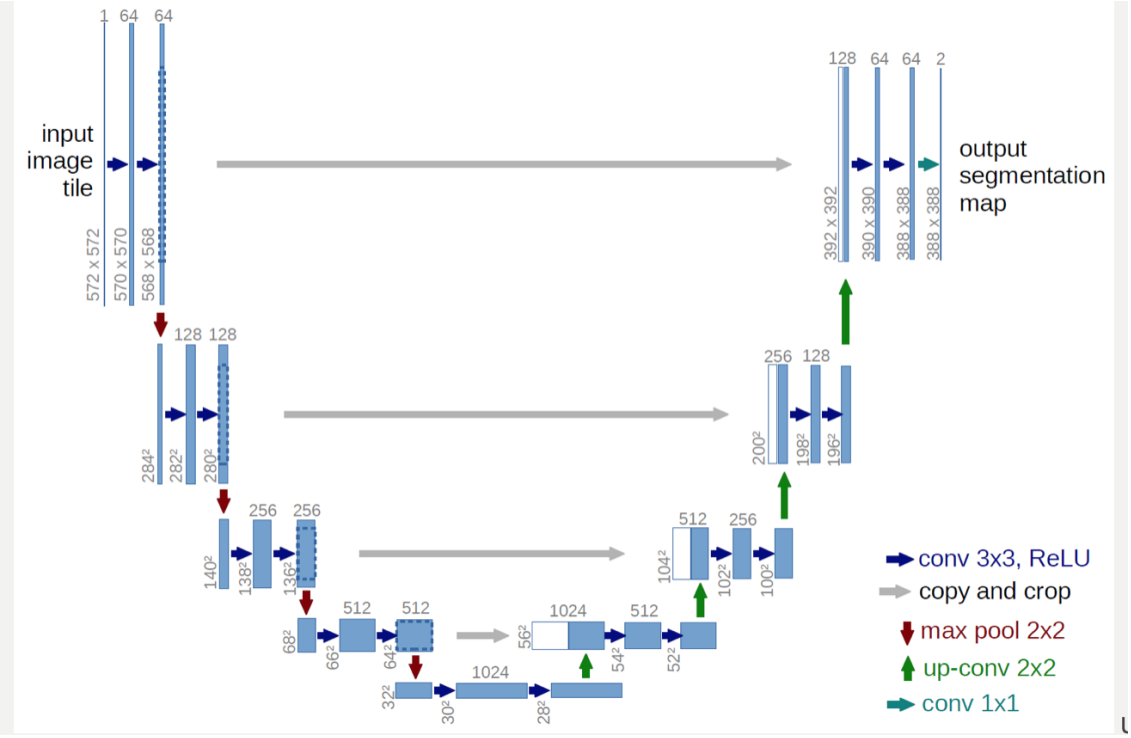


Figure 2: U Net Architecture

## 2 Visualisation of Examples

Here, we are going to visualise some examples of images and matching labels.


Figure 3: Images and Matching Labels

We can clearly see that the image and its matching labels, in the above figure. The original images are in RGB while their masks are binary (black and white).

# 3 Building the U Net model

**Definition of a choice of separation between training base and test base :** We will work on 64x64 size images. This part consists in the realization of a function allowing to select the central quarter of the slices containing the liver. We'll see the architecture in detail as follows :

Encoder part:

- Two convolution layers
- A Pooling layer (maxPool)
- A ReLu activation function

```
# Encoder
for filters in n_channels:
    x = Conv2D(filters, k_size, padding="same", activation="relu", kernel_initializer=k_init)(x)
    l_conv1.append(x) # (64, 64, 16), (32, 32, 32), (16, 16, 64), (8, 8, 128)

    x = Conv2D(filters, k_size, padding="same", activation="relu", kernel_initializer=k_init)(x)
    l_conv2.append(x) # (64, 64, 16), (32, 32, 32), (16, 16, 64), (8, 8, 128)
    output_enc.append(x)

    x = MaxPooling2D(pool_size=(2,2), strides=2, padding="same")(x)
    l_maxpool.append(x) # (32, 32, 16) , (16, 16, 32) , (8, 8, 64) , (4, 4, 128)
```

Figure 4: Encoder

Decoder part:

- A transposed convolution layer
- Two convolution layers
- A ReLu function

```
# Decoder
for i,filters in enumerate(n_channels[::-1]):
    x = Conv2DTranspose(filters, k_size, padding="same", activation="relu", kernel_initializer=k_init, strides=2)(x)
    l_tconv.append(x)
    x = concatenate([output_enc[3-i], l_tconv[i]], axis=-1)
    x = Conv2D(filters, k_size, padding="same", activation="relu", kernel_initializer=k_init)(x)
    x = Conv2D(filters, k_size, padding="same", activation="relu", kernel_initializer=k_init)(x)
```

Figure 5: Decoder

Because the deepest block doesn't have a maxpooling layer, so pick it out and process it individually like beneath, this layer exists between the encoder part and the decoder part.

```
# The deepest block has no pooling layer
x = Conv2D(256, k_size, padding="same", activation="relu", kernel_initializer=k_init)(x)
l_conv1.append(x)
x = Conv2D(256, k_size, padding="same", activation="relu", kernel_initializer=k_init)(x)
l_conv2.append(x)
```

Figure 6: The deepest block

**Metrics :** We have chosen 'Mean IOU' as the metric. This metric allows to measure if the prediction is correct compared to the ground truth for the location of objects in an image. The prediction is correct if IoU >= 0.5. The IoU allows to measure the similarity between the 2 boxes :

- The terrain truth box and the bounding box predicted by the model.
- IoU is calculated as the ratio between the overlapping area of the two boxes and the union area.

# 4 Training and testing

The figure below shows the compilation of our model. It contains 2,158,545 parameters.

| | | | |
|---|---|---|---|
| conv2d_10 (Conv2D) | (None, 8, 8, 128) | 295040 | concatenate[0][0] |
| conv2d_11 (Conv2D) | (None, 8, 8, 128) | 147584 | conv2d_10[0][0] |
| conv2d_transpose_1 (Conv2DTrans | (None, 16, 16, 64) | 73792 | conv2d_11[0][0] |
| concatenate_1 (Concatenate) | (None, 16, 16, 128) | 0 | conv2d_5[0][0]<br>conv2d_transpose_1[0][0] |
| conv2d_12 (Conv2D) | (None, 16, 16, 64) | 73792 | concatenate_1[0][0] |
| conv2d_13 (Conv2D) | (None, 16, 16, 64) | 36928 | conv2d_12[0][0] |
| conv2d_transpose_2 (Conv2DTrans | (None, 32, 32, 32) | 18464 | conv2d_13[0][0] |
| concatenate_2 (Concatenate) | (None, 32, 32, 64) | 0 | conv2d_3[0][0]<br>conv2d_transpose_2[0][0] |
| conv2d_14 (Conv2D) | (None, 32, 32, 32) | 18464 | concatenate_2[0][0] |
| conv2d_15 (Conv2D) | (None, 32, 32, 32) | 9248 | conv2d_14[0][0] |
| conv2d_transpose_3 (Conv2DTrans | (None, 64, 64, 16) | 4624 | conv2d_15[0][0] |
| concatenate_3 (Concatenate) | (None, 64, 64, 32) | 0 | conv2d_1[0][0]<br>conv2d_transpose_3[0][0] |
| conv2d_16 (Conv2D) | (None, 64, 64, 16) | 4624 | concatenate_3[0][0] |
| conv2d_17 (Conv2D) | (None, 64, 64, 16) | 2320 | conv2d_16[0][0] |
| conv2d_transpose_4 (Conv2DTrans | (None, 64, 64, 1) | 145 | conv2d_17[0][0] |

Total params: 2,158,833
Trainable params: 2,158,833
Non-trainable params: 0

Figure 7: Model Architecture Compilation

After training, we can draw the curves of the history of the loss and the metric:

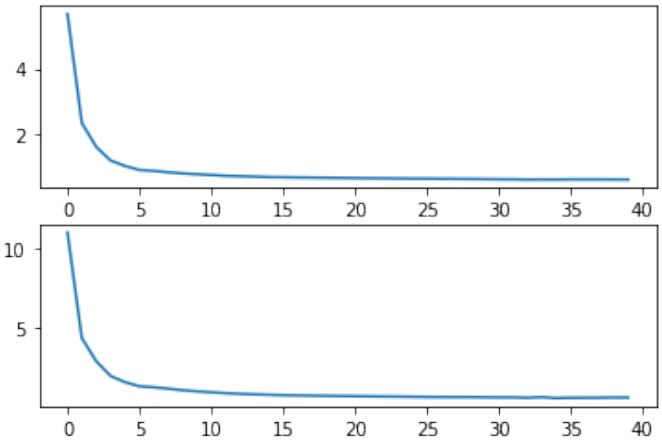


Figure 8: History of the loss and the metric of training data

The images in the test folder don't have their truth terrain, so we split our original train data into new train data(40 images) and test data(10 images) at the very beginning.

```
# We split the original train data into new train data(40 images) and test data(10 images)
X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train, test_size=0.2, random_state=0)
```

Figure 9: Data split

And then, we make predictions with the test images and visualize the predictions

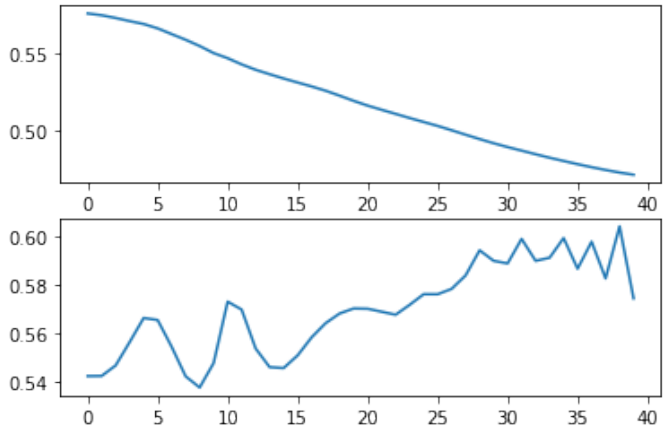


Figure 10: History of the loss and the metric of testing data

In order to see how well our model works we visualise five test images in the 1st row, and then their masks in the 2nd row, the 3rd row is our prediction. According to the results we can conclude that our model can find and split the targets, however, it seems to be not very precise, especially when there are many and small cells in a limited region.

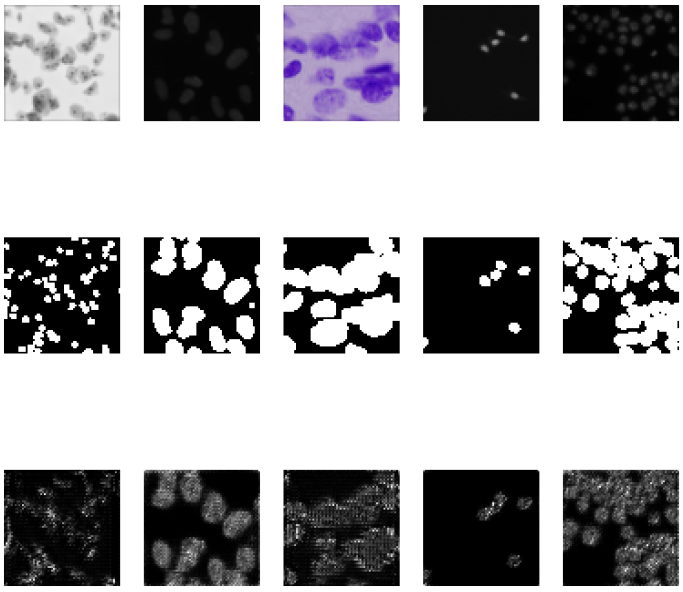

Figure 11: Comparison among images, masks and predictions

## 5 Comparison and Improvement

As we conclude above, we can see the model u net is enough complex which contains so many parameters, but in our situation, the train set is not so big. So it will absolutely cause the problem of over-fitting, more specifically, the performance of train set is better than the test set. So it's necessary to take some measures in order to improve the performance in the test set. And we chose the method of data augmentation to improve its precision.

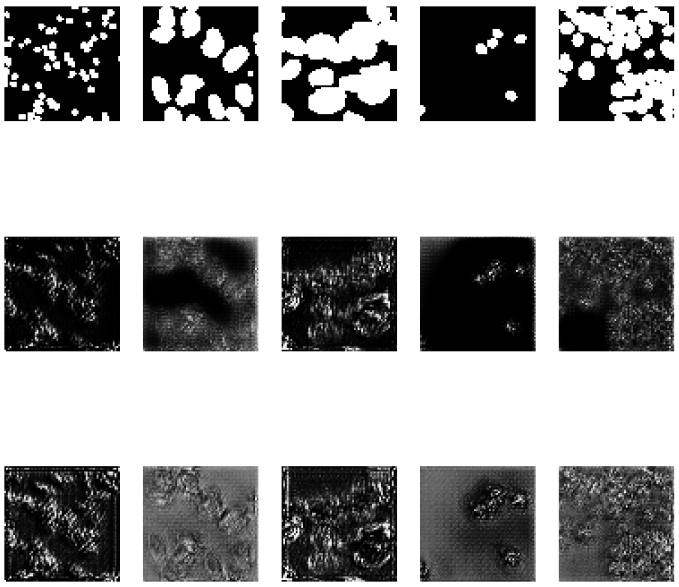

Figure 12: Comparison among masks and predictions

In the figure above, the 1st row represents the mask (or truth terrain), the 2nd row is our prediction without data augmentation and the 3rd row is the prediction with data augmentation. We find it clearer in the 3rd row than the 2nd row, which means data augmentation does improve the precision of our prediction.