The purpose of this laboratory is to develop a segmentation model by following one of the most efficient segmentation networks named as U-Net. The architecture of this network looks like a ‘U’ shape. This model consists three sections: Contraction, Bottleneck, and Expansion. One of the advantages of this model is that it is a fully convolutional network. Therefore, it is capable to maintain the spatial information. This architecture contains 4 convolutional blocks at the contraction path, 4 convolutional blocks at the expansion path and one convolutional layer at the bottleneck.

Task0) Developing a modular segmentation pipeline in a way that you prepare separate functions for different steps such as image loader, mask loader, augmentation generator, similarity metrics, and the model. Therefore, your main file should contain only few lines for calling the functions and setting the parameters. Please note for the segmentation tasks, for each image there will be corresponding target mask. For developing a 2D model architecture follow this instruction:

Each convolutional block at contraction path should entail two convolutional layers with the same number of feature maps and a max pooling layer. As the model go deeper, the number of feature maps at each block should be doubled. The presence of batch normalization and dropout layers should be optional. If you increase the number of feature maps of each block, then the convolutional layer at the bottleneck layer should be exactly 16 times larger than the first block.

The expansion path should be designed symmetrical with respect to the contraction path. Each block at this path starts with a transpose convolution follows by a concatenation layer, optional drop out, and a convolutional block with the same structure as what you employed for the expansion path.

Task1a) Lung segmentation in chest X-ray images: Instantiate your pipeline with the following parameters: # of filters at first layer (base)=16; image size=256; batch size=8; learning rate = 0.0001; drop out rate=0.5; batch normalization = True; Metric = Dice Coefficient; no data augumentation; and loss function = binary cross entropy. Reading the ‘X_ray’ images (images and masks), randomly shuffle them and assign 80 percent of them for training and the rest of 20 percent for validation. Train the model for 150 epochs and save your final results.

```
Path = ‘/Lab1/Lab3/X_ray/’
```

Task1b) keep all the parameters from previous exercise the same, but only change the loss function to “Dice loss” and repeat the exercise.

Is there any difference between model performance over validation set when you changed the loss functions? Discuss it.

Task2) Repeat the tasks 1a and 1b without applying the batch normalization technique. Does it effect the model performance? What about the learning curves?

Taks3) In tasks 1,2 you already worked with 4 different settings. Choose the setting with the best performance and set the # of filters at the first layer (base) as 32 and train the model for 150 epochs. Does it change the segmentation accuracy? In general, how do you choose the number of feature maps?

Taks4) Similar to task3, employ the setting with the best performance and apply the augmentation technique on both of images and masks: rotation range=10; width and height shift ranges=0.1; zoom range = 0.2 and horizontal flip. Could data augmentation improve the model performance? What about generalization power?

Task5a) Lung segmentation in CT images: Reuse the exact settings of the task1 for "CT" segmentation. Which of the X-ray or CT images yielded more accurate segmentation results? Why?

Task5b) Evaluating the segmentation performance is often done by Dice coefficient; however, it is quite essential to assess the level of false positives and false negatives too. Repeat the task 5a by including data augmentation technique (similar to task4). Moreover, implement "precision" and "recall" metrics and along with "dice coefficient" apply all of them on your model. (Metric = [dice_coef, precision, recall]). Interpret the observed values of these three metrics over the validation set.

Task6) As you might already noticed, the labels of the left and right lungs in the CT masks are not the same values. We can consider the two lungs as two different organs and perform a multi-organ segmentation. Modify your model and adapt it for a multi-organ segmentation task and segment the left and right lungs separately in one framework.

Task7) Brain tumor segmentation in MRI images: In contrast to the lungs that entails specific shapes, appearances, and intensity patterns, brain tumors can be presented in a variety of shape, appearance, texture and intensity patterns. More importantly, they can be presented in any regions inside the brain therefore they do not have a fixed location. In this exercise you are asked to optimize your model by tuning the hyperparameters for the challenging task of brain tumor segmentation. Please note the original size of the brain MRI are 240 by 240.

Bonus Task) In the previous tasks, you gained experience and knowledge to develop the modular 2D segmentation models. In this exercise, you should extend your model to a 3D (volumetric) segmentation model for the task of lung segmentation in CT images. The model extension will be not be difficult; however, the size of CT images is $(512 \times 512 \times x)$ where variable x represents the number of image slices (depths). As x is not a fixed number, the input shape of model will be variable. Different strategies have been developed to tackle this issue. One conventional approach is that instead of using the whole image size, extracting fixed patch sizes from the image and feed the models with patches. This approach has another advantage that helps to significantly decease the size of the model. Another approach is to resample all the data into a fixed size and then process the resampled data. For this task and for simplicity, you should isotropically resample all the input volumetric images into a fixed size. Another important step in data preparation is to use a proper intensity window for CT images:

```
def normalize(image):

    image = (image - MIN_BOUND) / (MAX_BOUND - MIN_BOUND)
    image[image>1] = 1.
    image[image<0] = 0.
    return image
MIN_BOUND = -1000.0
MAX_BOUND = 500.0
```

You can employ the SimpleITK libraries for reading and processing the volumetric images. Report your segmentation accuracy with the following three metrics: Dice, Precision, and Recall.

Good luck.