Brief

Write the architecture of a SimpleUNet following the diagram shown during the lecture/in GitHub:

Encoder: Define a convolutional block (conv_block) that will be repeated in each layer of the encoder. Define as well a maxpooling block (maxpool_block) that will also be repeated when necessary during the encoding (downsampling path). Hint: use torch.nn.Conv2d and remember you need to add by hand the activation function (use ReLU).

Middle: define the middle or bottleneck part of the architecture.

Decoder: Define the transposed convolutional blocks (transposed_block). Hint: use nn.ConvTranspose2d.

Final layer: Define the final convolutional layer.

Forward: Remember you need to define the forward pass of your class, with how each block will be called.

In order to aid the optimisation process, include batch normalisation (nn.BatchNorm2d) and 50% dropout (nn.Dropout2d(0.5)). Where do think they should be placed?

Note: There are different ways to build this architecture, here we will use PyTorch as we will test that we can load the pre-trained model provided.

```
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
import matplotlib.pyplot as plt
!pip install torchmetrics
import torchmetrics
from torchmetrics.classification import BinaryAccuracy
import numpy as np
from torchsummary import summary
import cv2 # optional for the bonus questions
```

```
Collecting torchmetrics
  Downloading torchmetrics-1.3.1-py3-none-any.whl (840 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 840.4/840.4 kB 5.5 MB/s eta 0:00:00
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (1.25.2)
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (23.2)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (2.1.0+cu121
Collecting lightning-utilities>=0.8.0 (from torchmetrics)
  Downloading lightning_utilities-0.10.1-py3-none-any.whl (24 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->t
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (3
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (1.12
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (3
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (3.1
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (202
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetric
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.10.0->t
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.10.0->torch
Installing collected packages: lightning-utilities, torchmetrics
Successfully installed lightning-utilities-0.10.1 torchmetrics-1.3.1
```

Fill the empty spaces to define the architecture of the UNet as the image provided in the problem sheet (call it SimpleUNet)

```
class SimpleUNet(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.conv1 = self.conv_block(in_channels, 16, 3, 1, 1)
        self.maxpool1 = self.maxpool_block(2, 2, 0)
        self.conv2 = self.conv_block(16, 32, 3, 1, 1)
        self.maxpool2 = self.maxpool_block(2, 2, 0)
        self.conv3 = self.conv_block(32, 64, 3, 1, 1)
        self.maxpool3 = self.maxpool_block(2, 2, 0)

        self.middle = self.conv_block(64, 128, 3, 1, 1)

        self.upsample3 = self.transposed_block(128, 64, 3, 2, 1, 1)
        self.upconv3 = self.conv_block(128, 64, 3, 1, 1)
        self.upsample2 = self.transposed_block(64, 32, 3, 2, 1, 1)
        self.upconv2 = self.conv_block(64, 32, 3, 1, 1)

        self.upsample1 = self.transposed_block(32, 16, 3, 2, 1, 1)
        self.upconv1 = self.conv_block(32, 16, 3, 1, 1)

        self.final = self.final_layer(16, 1, 1, 1, 0)

    def conv_block(self, in_channels, out_channels, kernel_size, stride, padding):
       convolution = nn.Sequential(
                    nn.Conv2d(in_channels, out_channels, kernel_size=kernel_size, stride=stride, padding=padding),
```

```
                    nn.BatchNorm2d(out_channels),
                    nn.ReLU(inplace=True),
                    nn.Conv2d(out_channels, out_channels, kernel_size=kernel_size, stride=stride, padding=padding),
                    nn.BatchNorm2d(out_channels),
                    nn.ReLU(inplace=True))

        return convolution

    def maxpool_block(self, kernel_size, stride, padding):
        maxpool = nn.Sequential(nn.MaxPool2d(kernel_size=kernel_size, stride=stride, padding=padding),
               nn.Dropout2d(0.5))
        return maxpool

    def transposed_block(self, in_channels, out_channels, kernel_size, stride, padding, output_padding):
        transposed = torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size=kernel_size, stride=stride,
                                              padding=padding, output_padding=output_padding)
        return transposed

    def final_layer(self, in_channels, out_channels, kernel_size, stride, padding):
        final = nn.Conv2d(in_channels, out_channels, kernel_size=kernel_size, stride=stride, padding=padding)
        return final

    def forward(self, x):
        # downsampling part
        conv1 = self.conv1(x)
        maxpool1 = self.maxpool1(conv1)
        conv2 = self.conv2(maxpool1)
        maxpool2 = self.maxpool2(conv2)
        conv3 = self.conv3(maxpool2)
        maxpool3 = self.maxpool3(conv3)

        # middle part
        middle = self.middle(maxpool3)

        # upsampling part
        upsample3 = self.upsample3(middle)
        upconv3 = self.upconv3(torch.cat([upsample3, conv3], 1))
        upsample2 = self.upsample2(upconv3)
        upconv2 = self.upconv2(torch.cat([upsample2, conv2], 1))
        upsample1 = self.upsample1(upconv2)
        upconv1 = self.upconv1(torch.cat([upsample1, conv1], 1))

        final_layer = self.final(upconv1)

        return final_layer
```

Now load the pre-trained model and test if the implementation above is correct by printing the summary of the model.

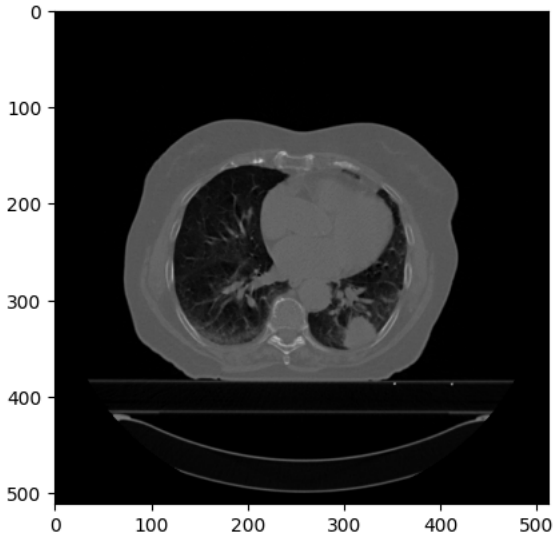
```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = SimpleUNet(in_channels=1, out_channels=1)
model.load_state_dict(torch.load('SimpleUNet_v3.pt', map_location=torch.device(device)))
summary(model, input_size=(1, 512, 512))
```

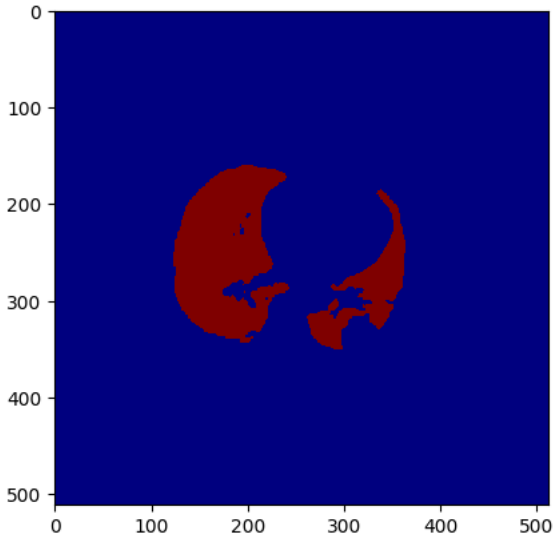
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 512, 512]             160
       BatchNorm2d-2         [-1, 16, 512, 512]              32
              ReLU-3         [-1, 16, 512, 512]               0
            Conv2d-4         [-1, 16, 512, 512]           2,320
       BatchNorm2d-5         [-1, 16, 512, 512]              32
              ReLU-6         [-1, 16, 512, 512]               0
         MaxPool2d-7         [-1, 16, 256, 256]               0
         Dropout2d-8         [-1, 16, 256, 256]               0
            Conv2d-9         [-1, 32, 256, 256]           4,640
      BatchNorm2d-10         [-1, 32, 256, 256]              64
             ReLU-11         [-1, 32, 256, 256]               0
           Conv2d-12         [-1, 32, 256, 256]           9,248
      BatchNorm2d-13         [-1, 32, 256, 256]              64
             ReLU-14         [-1, 32, 256, 256]               0
        MaxPool2d-15         [-1, 32, 128, 128]               0
        Dropout2d-16         [-1, 32, 128, 128]               0
           Conv2d-17         [-1, 64, 128, 128]          18,496
      BatchNorm2d-18         [-1, 64, 128, 128]             128
             ReLU-19         [-1, 64, 128, 128]               0
           Conv2d-20         [-1, 64, 128, 128]          36,928
      BatchNorm2d-21         [-1, 64, 128, 128]             128
             ReLU-22         [-1, 64, 128, 128]               0
        MaxPool2d-23           [-1, 64, 64, 64]               0
        Dropout2d-24           [-1, 64, 64, 64]               0
           Conv2d-25          [-1, 128, 64, 64]          73,856
      BatchNorm2d-26          [-1, 128, 64, 64]             256
             ReLU-27          [-1, 128, 64, 64]               0
```

```
            Conv2d-28          [-1, 128, 64, 64]         147,584
       BatchNorm2d-29          [-1, 128, 64, 64]             256
              ReLU-30          [-1, 128, 64, 64]               0
   ConvTranspose2d-31         [-1, 64, 128, 128]          73,792
            Conv2d-32         [-1, 64, 128, 128]          73,792
       BatchNorm2d-33         [-1, 64, 128, 128]             128
              ReLU-34         [-1, 64, 128, 128]               0
            Conv2d-35         [-1, 64, 128, 128]          36,928
       BatchNorm2d-36         [-1, 64, 128, 128]             128
              ReLU-37         [-1, 64, 128, 128]               0
   ConvTranspose2d-38         [-1, 32, 256, 256]          18,464
            Conv2d-39         [-1, 32, 256, 256]          18,464
       BatchNorm2d-40         [-1, 32, 256, 256]              64
              ReLU-41         [-1, 32, 256, 256]               0
            Conv2d-42         [-1, 32, 256, 256]           9,248
       BatchNorm2d-43         [-1, 32, 256, 256]              64
              ReLU-44         [-1, 32, 256, 256]               0
   ConvTranspose2d-45         [-1, 16, 512, 512]           4,624
            Conv2d-46         [-1, 16, 512, 512]           4,624
       BatchNorm2d-47         [-1, 16, 512, 512]              32
              ReLU-48         [-1, 16, 512, 512]               0
            Conv2d-49         [-1, 16, 512, 512]           2,320
       BatchNorm2d-50         [-1, 16, 512, 512]              32
              ReLU-51         [-1, 16, 512, 512]               0
            Conv2d-52          [-1, 1, 512, 512]              17
================================================================
Total params: 536,913
```

Test applying the model to a few 2D slices of the test case

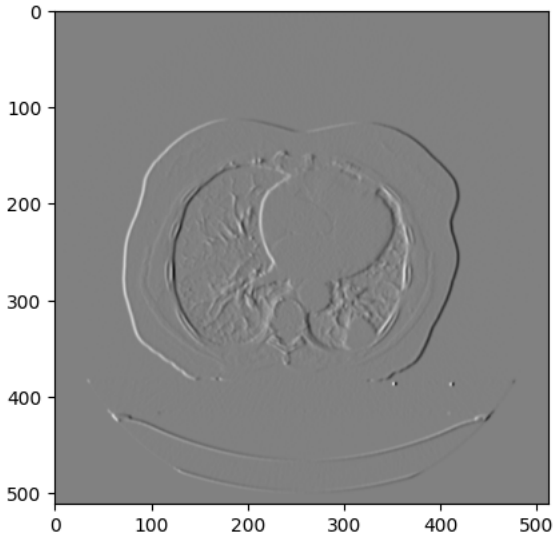
```
# Load the images to test
batch_size = 1
image_file = 'Case_010.npz'

image_array = (np.load(image_file))['images']
image_tensor = torch.Tensor(image_array)

# Model to evaluation mode
model.eval()
metric = BinaryAccuracy()
threshold = 0.5

# Choose and plot a 2D slice
slice_id = 42
curr_image = image_tensor[slice_id, :, :]
plt.imshow(curr_image, cmap='gray')
plt.show()

# We need to convert our input to 4D (including batch size) to compute predictions
curr_image = torch.unsqueeze(curr_image, 0)
curr_image = torch.unsqueeze(curr_image, 0)
curr_pred = model(curr_image)
curr_pred = torch.sigmoid(curr_pred)
curr_pred = (curr_pred > threshold)
curr_pred = curr_pred[0,0,:,:]
plt.imshow(curr_pred, cmap='jet', interpolation='none')
plt.show()
```

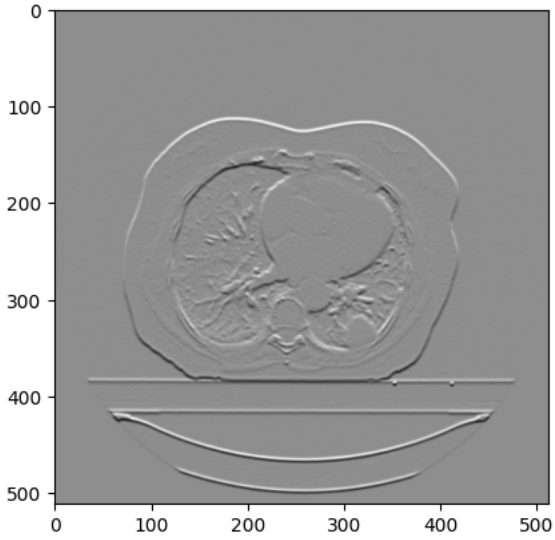




BONUS: Apply the Sobel algorithms using OpenCV

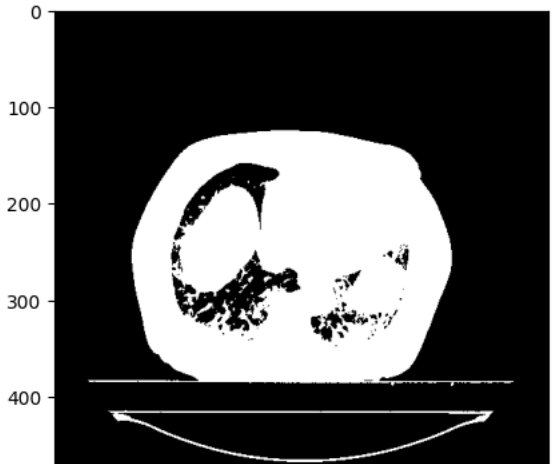
```
# Get again the slice as a numpy array
curr_slice = image_array[slice_id, :, :]

# Sobel Edge Detection
sobelx = cv2.Sobel(src=curr_slice, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5) # Sobel Edge Detection on the X axis
sobely = cv2.Sobel(src=curr_slice, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5) # Sobel Edge Detection on the Y axis
sobelxy = cv2.Sobel(src=curr_slice, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=5) # Combined X and Y Sobel Edge Detection
# Display Sobel Edge Detection Images
plt.imshow(sobelx, cmap='gray')
plt.show()
plt.imshow(sobely, cmap='gray')
plt.show()
plt.imshow(sobelxy, cmap='gray')
plt.show()
```





Compare to Thresholding results: binarisation and Otsu

```
image_file = 'Case_010.npz'
image_array = (np.load(image_file))['images']
slice_id = 32
curr_slice = image_array[slice_id, :, :]

# Binarisation:
# we will use the knowledge that air is at -1000 H.U.
# in this case we haven't converted to H.U. yet
curr_slice_threshold = curr_slice >400
plt.imshow(curr_slice_threshold, cmap='gray')
plt.show()

# Otsu, using OpenCV, and an uint8 version of the image
curr_slice_int = curr_slice.astype("uint8")
ret,image_otsu = cv2.threshold(curr_slice_int,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
plt.imshow(image_otsu, cmap='gray')
plt.show()
```


0
100
200
300
400