Deep Learning Project Documentation

Students: Anar Abiyev (W19RIG), Ramil Khalilli (EV4EER)
Course: Deep Learning.

Introduction

Image segmentation tasks are one of the most challenging and important problems in the field of AI. With the growing power of AI, these methods are now used in healthcare. Segmentation in healthcare means that the different parts of the body are separated into regions. This is quite a useful step in the process of illness detection. The data for this task contains heart Magnetic Resonance Images (MRI). The dataset has been organized as pairs of images and masks. The goal of the model is to be able to generate a mask for new MRI images.

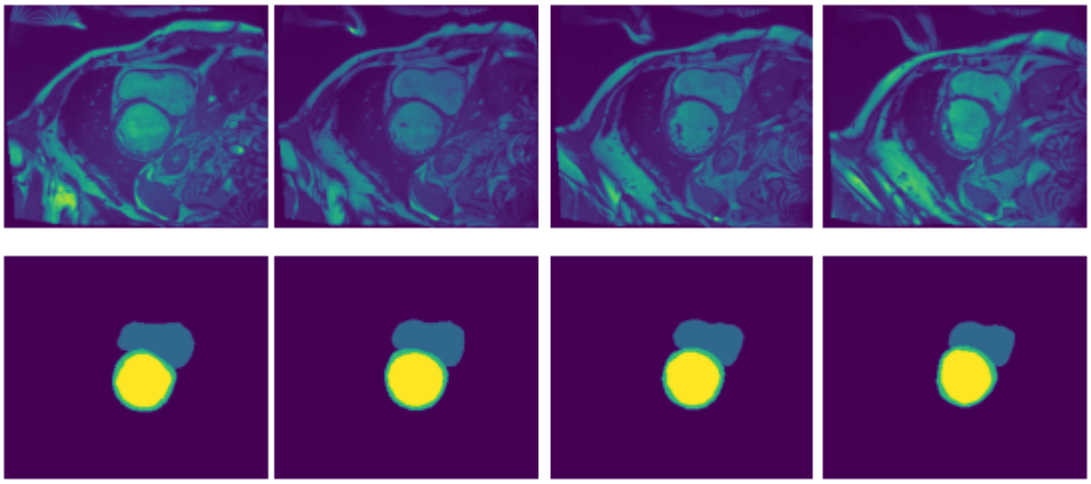

Figure 1. Example of dataset.

Methodology

The applied methodology for this task is to divide the problem into two sections: data preparation and modelling.
The dataset was given as separate .nii files for each patient. These files were read using relevant libraries of Python and stored as pixel arrays. To simplify the data processing and storing, the cloud of google drive has been used to integrate with google colab.
In data preparation, the MRI images have been separated into separate channels, because the model needs to get 3D arrays, not 4. Another issue was that the shapes of the images were different than one another. To solve this problem, padding was used, then images were resized to 256x256 pixels.
In the next data preparation step, the images were rotated 90 degrees to synthetically increase the size of the dataset. Here, both images and corresponding masks were rotated to keep the relationship between them. To summarize the data preparation process:

- Channel separation.
- Shape correction.
- Data augmentation.
- Saving dataset for modelling.

During modelling, the selected model structure is U-net. The U-Net architecture is a popular convolutional neural network (CNN) designed for semantic segmentation tasks, where the goal is to classify each pixel in an image into different classes. U-Net was originally developed for biomedical image segmentation but has since found applications in various domains. The key feature of U-Net is its U-shaped architecture, consisting of a contracting path (encoder) and an expansive path (decoder), with skip connections to preserve spatial information.

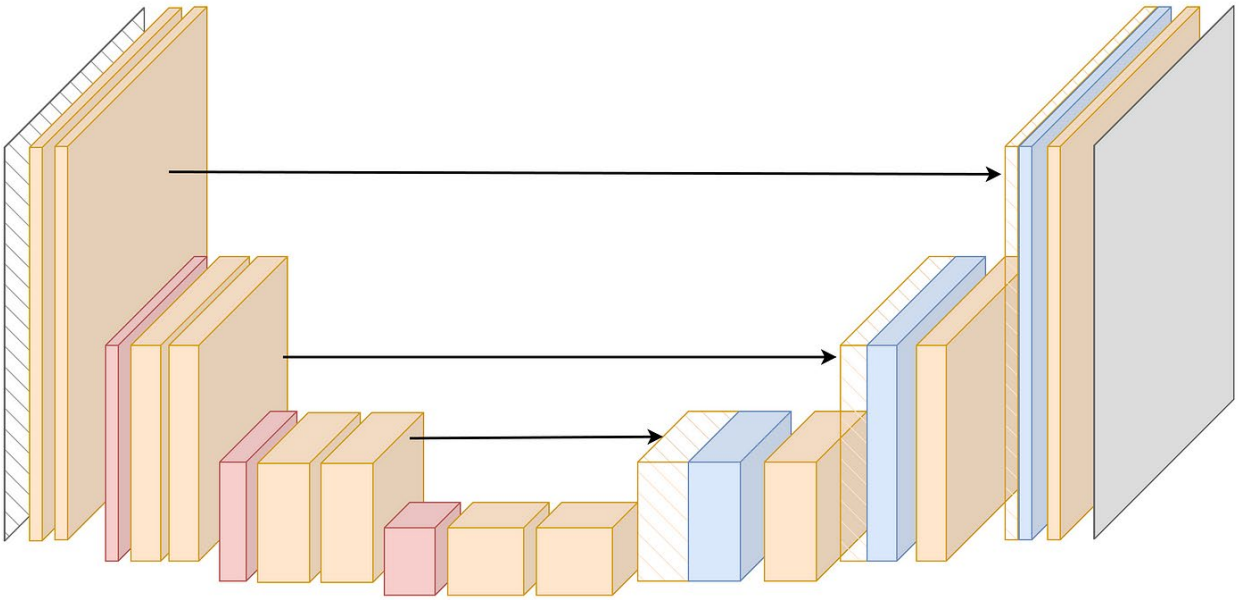

Figure 2. Representation of U-Net architecture.

The dataset was initially divided into train and test folders. The X value are images and y value are masks. At this point, the missing part is model structure to train with the dataset. For this purpose, three different U-net structures were tested. The difference between them is the number of convolutional filters in convolutional layers.

```
#Contraction path
c1 = tf.keras.layers.Conv2D(16, (3, 3),
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(16, (3, 3),
p1 = tf.keras.layers.MaxPooling2D((2, 2))

c2 = tf.keras.layers.Conv2D(32, (3, 3),
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(32, (3, 3),
p2 = tf.keras.layers.MaxPooling2D((2, 2))

c3 = tf.keras.layers.Conv2D(64, (3, 3),
c3 = tf.keras.layers.Dropout(0.2)(c3)
c3 = tf.keras.layers.Conv2D(64, (3, 3),
p3 = tf.keras.layers.MaxPooling2D((2, 2))

c4 = tf.keras.layers.Conv2D(128, (3, 3),
c4 = tf.keras.layers.Dropout(0.2)(c4)
c4 = tf.keras.layers.Conv2D(128, (3, 3),
p4 = tf.keras.layers.MaxPooling2D(pool_si

c5 = tf.keras.layers.Conv2D(256, (3, 3),
c5 = tf.keras.layers.Dropout(0.3)(c5)
c5 = tf.keras.layers.Conv2D(256, (3, 3),
```

```
#Expansive path
u6 = tf.keras.layers.Conv2DTranspose(128, (2, 2),
u6 = tf.keras.layers.concatenate([u6, c4])
c6 = tf.keras.layers.Conv2D(128, (3, 3),
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(128, (3, 3),

u7 = tf.keras.layers.Conv2DTranspose(64, (2, 2),
u7 = tf.keras.layers.concatenate([u7, c3])
c7 = tf.keras.layers.Conv2D(64, (3, 3),
c7 = tf.keras.layers.Dropout(0.2)(c7)
c7 = tf.keras.layers.Conv2D(64, (3, 3),

u8 = tf.keras.layers.Conv2DTranspose(32, (2, 2),
u8 = tf.keras.layers.concatenate([u8, c2])
c8 = tf.keras.layers.Conv2D(32, (3, 3),
c8 = tf.keras.layers.Dropout(0.1)(c8)
c8 = tf.keras.layers.Conv2D(32, (3, 3),

u9 = tf.keras.layers.Conv2DTranspose(16, (2, 2),
u9 = tf.keras.layers.concatenate([u9, c1], axis=3
c9 = tf.keras.layers.Conv2D(16, (3, 3),
c9 = tf.keras.layers.Dropout(0.1)(c9)
c9 = tf.keras.layers.Conv2D(16, (3, 3),
```

Figure 3. Model 1.

```
# Contraction path
c1 = tf.keras.layers.Conv2D(32, (3, 3),
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(32, (3, 3),
p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

c2 = tf.keras.layers.Conv2D(64, (3, 3),
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(64, (3, 3),
p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

c3 = tf.keras.layers.Conv2D(128, (3, 3),
c3 = tf.keras.layers.Dropout(0.2)(c3)
c3 = tf.keras.layers.Conv2D(128, (3, 3),
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(256, (3, 3),
c4 = tf.keras.layers.Dropout(0.2)(c4)
c4 = tf.keras.layers.Conv2D(256, (3, 3),
p4 = tf.keras.layers.MaxPooling2D(pool_size=(2,

c5 = tf.keras.layers.Conv2D(512, (3, 3),
c5 = tf.keras.layers.Dropout(0.3)(c5)
c5 = tf.keras.layers.Conv2D(512, (3, 3),
```

```
# Expansive path
u6 = tf.keras.layers.Conv2DTranspose(256, (2, 2),
u6 = tf.keras.layers.concatenate([u6, c4])
c6 = tf.keras.layers.Conv2D(256, (3, 3),
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(256, (3, 3),

u7 = tf.keras.layers.Conv2DTranspose(128, (2, 2),
u7 = tf.keras.layers.concatenate([u7, c3])
c7 = tf.keras.layers.Conv2D(128, (3, 3),
c7 = tf.keras.layers.Dropout(0.2)(c7)
c7 = tf.keras.layers.Conv2D(128, (3, 3),

u8 = tf.keras.layers.Conv2DTranspose(64, (2, 2),
u8 = tf.keras.layers.concatenate([u8, c2])
c8 = tf.keras.layers.Conv2D(64, (3, 3),
c8 = tf.keras.layers.Dropout(0.1)(c8)
c8 = tf.keras.layers.Conv2D(64, (3, 3),

u9 = tf.keras.layers.Conv2DTranspose(32, (2, 2),
u9 = tf.keras.layers.concatenate([u9, c1], axis=3)
c9 = tf.keras.layers.Conv2D(32, (3, 3),
c9 = tf.keras.layers.Dropout(0.1)(c9)
c9 = tf.keras.layers.Conv2D(32, (3, 3),
```

Figure 4. Model 2.

```
# Contraction path
c1 = tf.keras.layers.Conv2D(8, (3, 3),
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(8, (3, 3),
p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

c2 = tf.keras.layers.Conv2D(16, (3, 3),
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(16, (3, 3),
p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

c3 = tf.keras.layers.Conv2D(32, (3, 3),
c3 = tf.keras.layers.Dropout(0.2)(c3)
c3 = tf.keras.layers.Conv2D(32, (3, 3),
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(64, (3, 3),
c4 = tf.keras.layers.Dropout(0.2)(c4)
c4 = tf.keras.layers.Conv2D(64, (3, 3),
p4 = tf.keras.layers.MaxPooling2D(pool_size=(2,

c5 = tf.keras.layers.Conv2D(128, (3, 3),
c5 = tf.keras.layers.Dropout(0.3)(c5)
c5 = tf.keras.layers.Conv2D(128, (3, 3),
```

```
# Expansive path
u6 = tf.keras.layers.Conv2DTranspose(64, (2, 2),
u6 = tf.keras.layers.concatenate([u6, c4])
c6 = tf.keras.layers.Conv2D(64, (3, 3),
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(64, (3, 3),

u7 = tf.keras.layers.Conv2DTranspose(32, (2, 2),
u7 = tf.keras.layers.concatenate([u7, c3])
c7 = tf.keras.layers.Conv2D(32, (3, 3),
c7 = tf.keras.layers.Dropout(0.2)(c7)
c7 = tf.keras.layers.Conv2D(32, (3, 3),

u8 = tf.keras.layers.Conv2DTranspose(16, (2, 2),
u8 = tf.keras.layers.concatenate([u8, c2])
c8 = tf.keras.layers.Conv2D(16, (3, 3),
c8 = tf.keras.layers.Dropout(0.1)(c8)
c8 = tf.keras.layers.Conv2D(16, (3, 3),

u9 = tf.keras.layers.Conv2DTranspose(8, (2, 2),
u9 = tf.keras.layers.concatenate([u9, c1], axis=3)
c9 = tf.keras.layers.Conv2D(8, (3, 3),
c9 = tf.keras.layers.Dropout(0.1)(c9)
c9 = tf.keras.layers.Conv2D(8, (3, 3),
```

Figure 5. Model 3.

Results

To measure the results, the method of Dice Coefficient has been used. Dice value gets values between 0 and 1 and illustrates how successful the segmentation mask has been generated. The masks have been generated on both train and test datasets and dice coefficients have been calculated for both cases.

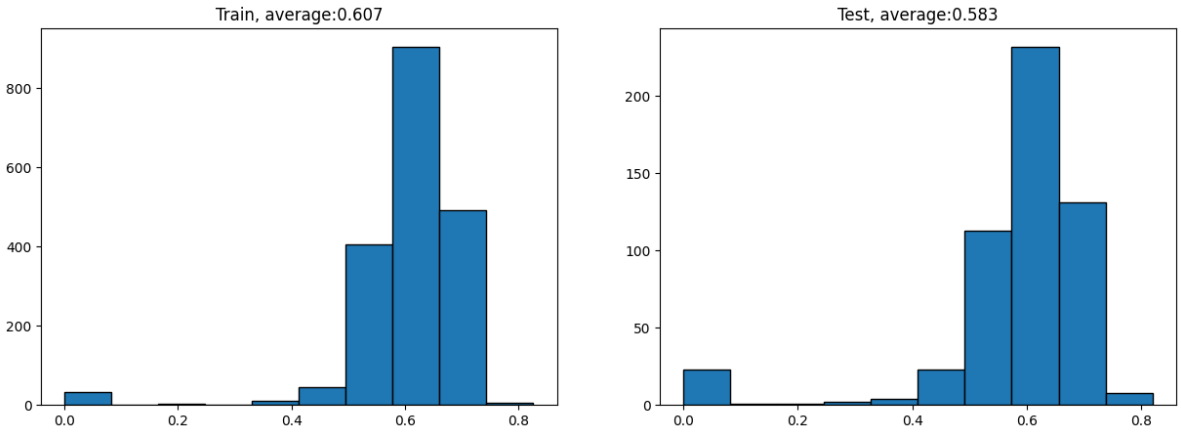

Figure 6. Results of model 1.

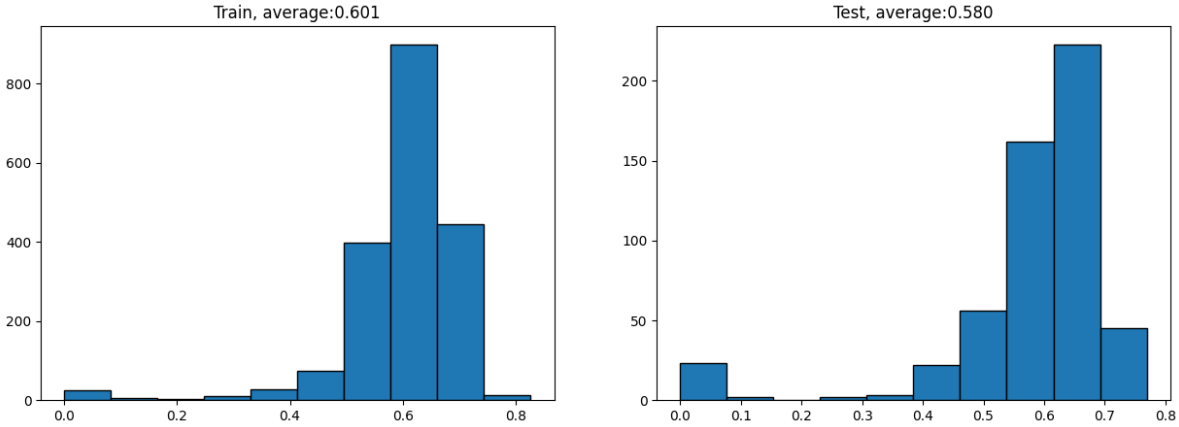

Figure 7. Results of model 2.

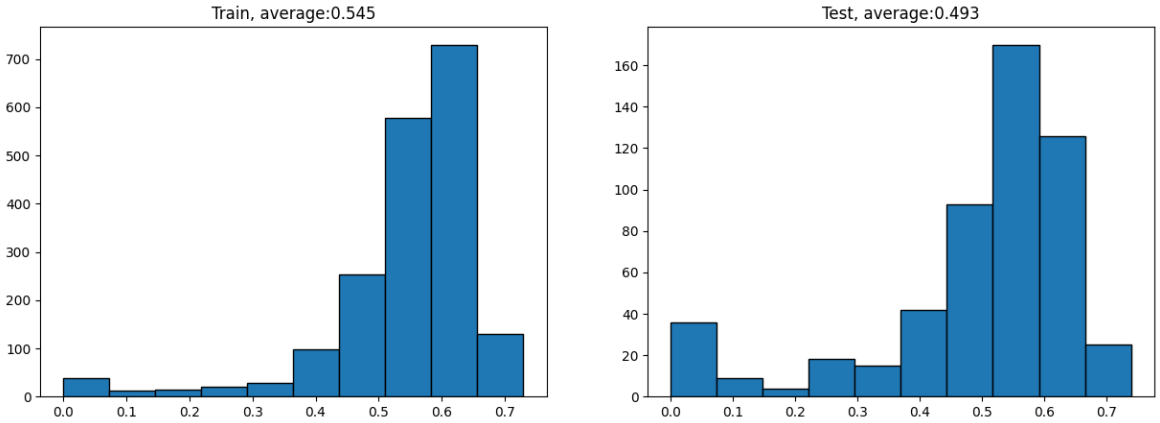

Figure 8. Results of model 3.

The figures below are random data samples from the predictions of the model, in the order of image, ground truth and prediction.

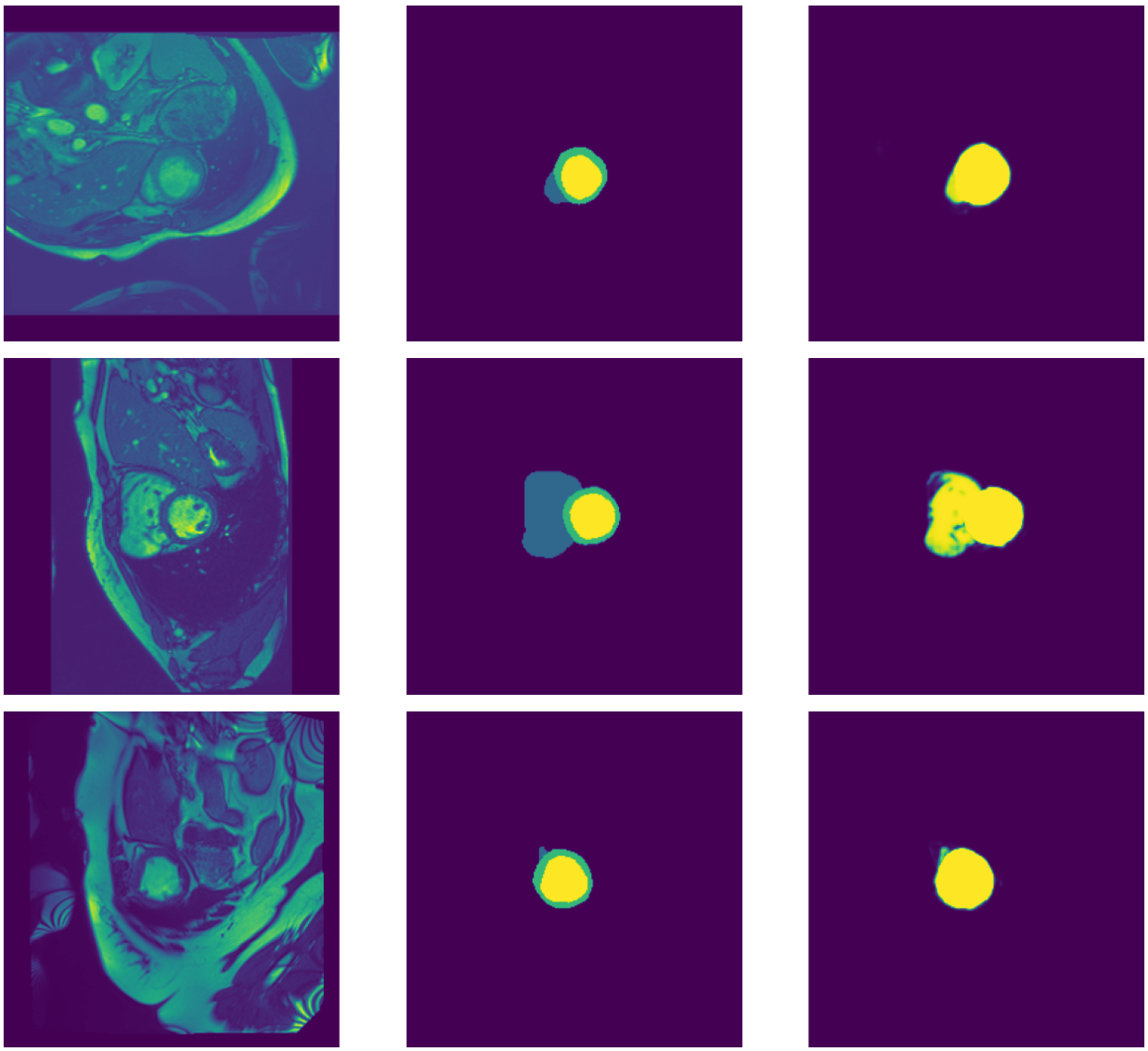

Figure 9. Samples of prediction.

Conclusion

To sum up, the segmentation problem was a challenging and interesting topic. This helped us to learn about image data preparation techniques likewise augmentation. After getting done with data, the modelling phase was a great way to use U-Net structure, which is a well-known and useful method to know about.
The results were just over average with an average dice coefficient of 0.6. The model is successful in finding the yellow areas of the masks, but it is not good enough to generalize the surrounding green area.