# RADDI 514 Assignment 4

## • U-net

The field of Deep Learning has recently exploded in popularity with the ease of access to large, public training datasets. In the case of medical image segmentation, Convolutional Neural Networks (CNNs) show promise as a logical evolution of traditional automated segmentation methods, because of their robustness and continuous improvement with larger training data.

U-Net is a CNN architecture designed for biomedical image segmentation tasks. Introduced by Ronneberger, et al in 2015, U-Net is characterized by a U-shaped architecture with an encoding path for capturing context and a symmetric decoding path for precise localization. The network is adept at handling tasks where the input and output have a one-to-one correspondence, such as segmenting structures in medical images. Its architecture enables the preservation of spatial information through bridged connections that concatenate feature maps from the encoding path to the decoding path, enabling the network to maintain fine-grained details during segmentation.

This report aims to segment the left ventricle by utilizing a U-net model. The model will be trained and evaluated using 60 training and 20 test images.

## • Method and Model Parameters

First, the images are normalized using the 5th and 95th percentile values, after which they are cropped from the center into 128*128 pixels to focus on the region of interest. Naturally, the same cropping process is applied to the ground truth as well. The 60 training data is then split further into 48 training samples and 12 (20%) validation samples. The U-net itself has 4 encoding convolutional blocks, 1 middle convolutional block, and 4 decoding convolutional blocks which have the feature maps concatenated with their corresponding encoder feature maps. As this segmentation has only two possible classes (foreground and background), the sigmoid activation function is used to classify each pixel into these categories. The output of the model is a probability function that shows how much each pixel is likely to be in the foreground or background.

Training is performed with 100 epochs, batch size of 4, and using the "Adam" optimizer with a 0.0005 learning rate. The dice coefficient is used as the loss function for training (1-Dice) and to monitor validation performance. Once training has finished, the weights and parameters with the best validation dice coefficient are saved to perform testing.

The test data go through the same normalization and cropping process as training data. The model predictions are computed into binary masks using a 0.5 threshold value. Next, these predicted masks are compared to the ground truth using Precision, Recall, Dice Coefficient and Jaccard Index (IoU).

## • Evaluation

The table below shows the model's average performance with 20 test data in four evaluations metrics:

| | **Precision** | **Recall** | **Dice Coefficient** | **Jaccard Index** |
|---|---|---|---|---|
| **Mean** | 0.7194 | 0.8272 | 0.7496 | 0.6372 |
| **STD** | 0.2540 | 0.2398 | 0.2169 | 0.2233 |

The training was completed in 39s 843ms. Prediction on 20 test data took 1s 72ms.

The following images compare the worst, 50th percentile and best segmentation outputs of the model:

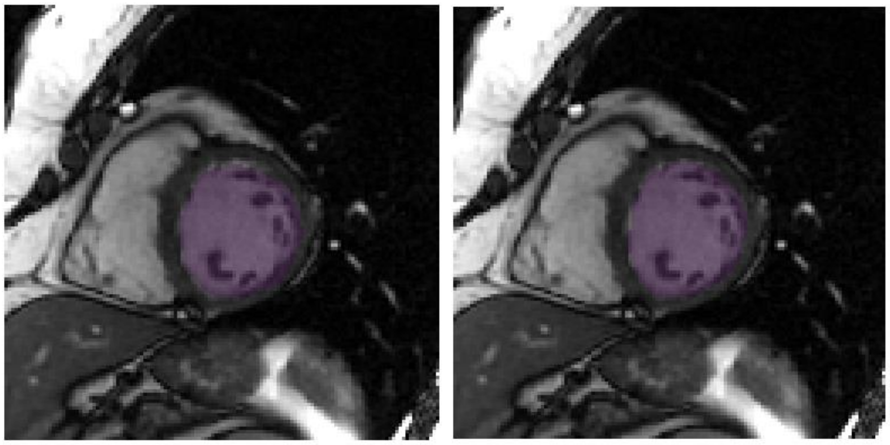

*Figure 1: Best performing output (right) and ground truth (left). The high overlap between the two results in 0.966 dice score.*

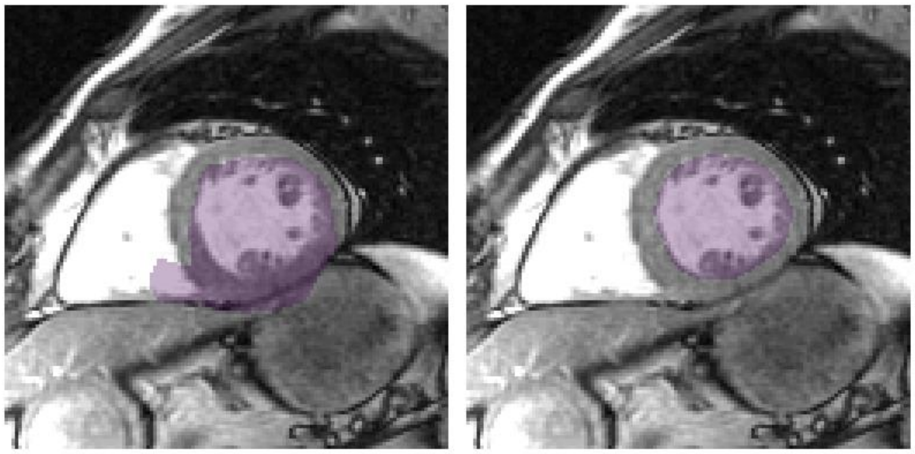

*Figure 2: 50th percentile result (right) and ground truth (left). The two masks largely overlap, but due to mislabeled sections on the prediction mask, dice score lowers to 0.7731*

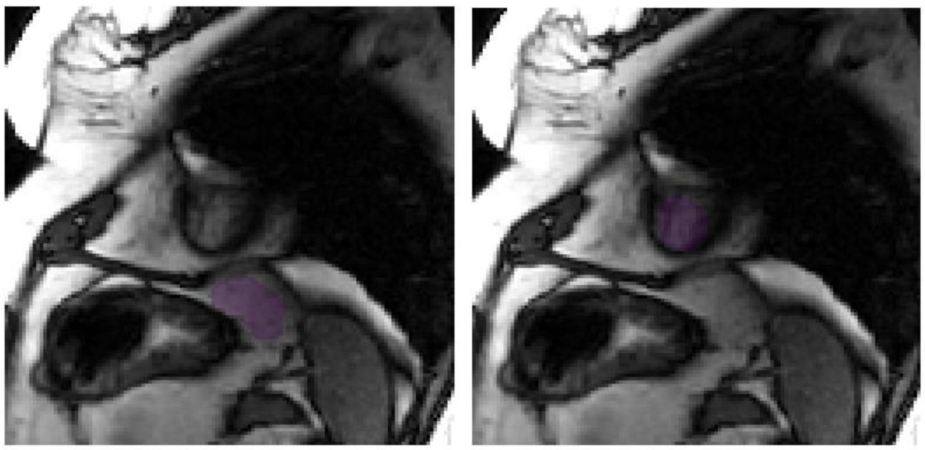

*Figure 3: The worst performing result of the model (right) and ground truth (left). The prediction is completely mislabeled, having no overlap with ground truth, leading to a dice score of 0*

- ## Source Code

***Training:***

```
#%%
from keras.models import Model
from keras.layers import Input, concatenate, Conv2D, MaxPooling2D,
Conv2DTranspose
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras import backend as K
from keras.callbacks import TensorBoard
from IPython.display import clear_output

import pydicom
import skimage.io
import glob
import os
import numpy as np
import matplotlib.pyplot as plt
#%%
# Display code for image and label
def display_im_label(img, lbl, titlef=""): # display the MR images along with
ground truth labels

    mask = np.ma.masked_where(lbl == 0, lbl)
    clear_output(wait=True) # clear figure
    plt.figure(figsize=(5,5))
    # show MR image
    plt.imshow(img, cmap='gray')
    plt.imshow(mask, alpha=0.3)
    plt.axis('off')

    plt.pause(0.01)
#%%
# Normalize image using 5th and 95th percentile values
def im_normalize(img):
    im_min, im_max = np.percentile(img,[5,95])
    return np.clip(np.array((img-im_min)/(im_max-im_min), dtype=np.float32),
0.0, 1.0)
#%%
# Perform image cropping from center
def crop_center(img, lbl, crx, cry):
    y, x = img.shape
    sx = x//2-(crx//2)
    sy = y//2-(cry//2)
    return img[sy:sy+cry,sx:sx+crx], lbl[sy:sy+cry,sx:sx+crx]
#%%
# Load images and labels
train_dir = os.path.join('lv_deeplearning','training')
SZ = 128 #Size
fimg_names = glob.glob(os.path.join(train_dir,'*.dcm'))

train_img, train_lbl = [], []
for fimg in fimg_names:
    ds = pydicom.read_file(fimg)
    lbl = skimage.io.imread("{}.png".format(fimg[:-4]))
    img = im_normalize(ds.pixel_array)
    img, lbl = crop_center(img,lbl,SZ,SZ)

    train_img.append(img)
    train_lbl.append(lbl)

    display_im_label(img,lbl)
#%%
# Set train image and label dimensions to 4 to be compatible with keras
train_img = np.array(train_img)[...,np.newaxis]
train_lbl = (np.array(train_lbl)[...,np.newaxis]>0).astype(np.float32)
#%%
# Define loss function and evaluation metric

eps = 1e-5

def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + eps) / (K.sum(y_true_f) + K.sum(y_pred_f) +
eps)

def dice_coef_loss(y_true, y_pred):
    return 1.0-dice_coef(y_true, y_pred)
#%%
# UNet neural network model definition
def get_unet():
    inputs = Input((SZ, SZ, 1))
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool3)
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool4)
    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv5)

    up6 = concatenate([Conv2DTranspose(256, (2, 2), strides=(2, 2),
padding='same')(conv5), conv4], axis=3)
    conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(up6)
    conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv6)

    up7 = concatenate([Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(conv6), conv3], axis=3)
    conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(up7)
    conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv7)

    up8 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(conv7), conv2], axis=3)
    conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(up8)
    conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv8)

    up9 = concatenate([Conv2DTranspose(32, (2, 2), strides=(2, 2),
padding='same')(conv8), conv1], axis=3)
    conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(up9)
    conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv9)

    conv10 = Conv2D(1, (1, 1), activation='sigmoid')(conv9)

    model = Model(inputs=[inputs], outputs=[conv10])

    return model
#%%
# Set neural network hyperparameters
N_EPOCHS = 100
LEARN_R = 5e-4
BATCH_SZ = 4

model = get_unet()
model.compile(optimizer=Adam(learning_rate=LEARN_R), loss=dice_coef_loss,
metrics=[dice_coef])
#%%
# Create neural network model and print its details
model.summary()
#%%
# Perform neural network training
print("train img", train_img.shape, type(train_img[0,0,0,0]),
np.min(train_img), np.max(train_img))
print("train lbl", train_lbl.shape, type(train_lbl[0,0,0,0]),
np.min(train_lbl), np.max(train_lbl))
print("Training data loaded successfully.\n")

print("Loading network...")
model_checkpoint = ModelCheckpoint(os.path.join('trained_models',
'trained_unet.h5'), monitor='val_loss', save_best_only=True)
tbCallBack = TensorBoard(log_dir=os.path.join('trained_models', 'Graph',
"train_ep{:04d}_lr{}_bsz_{:02d}".format(N_EPOCHS, LEARN_R, BATCH_SZ)),
write_graph=False, write_images=False)
print("Network loaded successfully.")

print("Starting network training ...")
model.fit(train_img, train_lbl, batch_size=BATCH_SZ, epochs=N_EPOCHS,
verbose=1, validation_split=0.2, callbacks=[model_checkpoint,tbCallBack])
print("Network trained successfully.")

K.clear_session()

print("done.")
#%%
```

***Prediction:***

```
#%%
from keras.models import Model
from keras.layers import Input, concatenate, Conv2D, MaxPooling2D,
Conv2DTranspose
from keras.optimizers import Adam
from keras import backend as K
from IPython.display import clear_output

import pydicom
from skimage.io import imread
import glob
import os
import numpy as np
import matplotlib.pyplot as plt

# Set neural network hyperparameters
N_EPOCHS = 100
LEARN_R = 5e-4
BATCH_SZ = 4
SZ = 128
#%%
def dice_coeff(groundtruth_mask, pred_mask): #Function to calculate Dice
coefficient
    intersect = np.sum(pred_mask*groundtruth_mask)
    total_sum = np.sum(pred_mask) + np.sum(groundtruth_mask)
    dice = np.mean(2*intersect/total_sum)
    return dice
def precision_score_(groundtruth_mask, pred_mask): #Function to calculate
Precision
    intersect = np.sum(pred_mask*groundtruth_mask)
    total_pixel_pred = np.sum(pred_mask)
    precision = np.mean(intersect/total_pixel_pred)
    return precision
def recall_score_(groundtruth_mask, pred_mask): #Function to calculate Recall
    intersect = np.sum(pred_mask*groundtruth_mask)
    total_pixel_truth = np.sum(groundtruth_mask)
    recall = np.mean(intersect/total_pixel_truth)
    return recall
def iou(groundtruth_mask, pred_mask): #Function to calculate Jaccard index
    intersect = np.sum(pred_mask*groundtruth_mask)
    union = np.sum(pred_mask) + np.sum(groundtruth_mask) - intersect
    iou = np.mean(intersect/union)
    return iou
#%%
# Normalize image using 5th and 95th percentile values
def im_normalize(img):
    im_min, im_max = np.percentile(img,[5,95])
    return np.clip(np.array((img-im_min)/(im_max-im_min), dtype=np.float32),
0.0, 1.0)
#%%
# Perform image cropping from center
def crop_center(img, lbl, crx, cry):
    y, x = img.shape
    sx = x//2-(crx//2)
    sy = y//2-(cry//2)
    return img[sy:sy+cry,sx:sx+crx], lbl[sy:sy+cry,sx:sx+crx]
#%%
# Define loss function and evaluation metric

eps = 1e-5

def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + eps) / (K.sum(y_true_f) + K.sum(y_pred_f) +
eps)

def dice_coef_loss(y_true, y_pred):
    return 1.0-dice_coef(y_true, y_pred)
#%%
# UNet neural network model definition
def get_unet():
    inputs = Input((SZ, SZ, 1))
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool3)
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool4)
    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv5)

    up6 = concatenate([Conv2DTranspose(256, (2, 2), strides=(2, 2),
padding='same')(conv5), conv4], axis=3)
    conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(up6)
    conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv6)

    up7 = concatenate([Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(conv6), conv3], axis=3)
    conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(up7)
    conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv7)

    up8 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(conv7), conv2], axis=3)
    conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(up8)
    conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv8)

    up9 = concatenate([Conv2DTranspose(32, (2, 2), strides=(2, 2),
padding='same')(conv8), conv1], axis=3)
```

```
    conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(up9)
    conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv9)

    conv10 = Conv2D(1, (1, 1), activation='sigmoid')(conv9)

    model = Model(inputs=[inputs], outputs=[conv10])

    model.compile(optimizer=Adam(learning_rate=LEARN_R), loss=dice_coef_loss,
metrics=[dice_coef])

    return model
#%%
# Load model and weights
model = get_unet()
model.load_weights(os.path.join('trained_models', 'trained_unet.h5'))
#%%
# Display MR image along with label
def display_im_label(img, lbl, titlef=""):

    mask = np.ma.masked_where(lbl == 0, lbl)
    clear_output(wait=True) # clear figure
    plt.figure(figsize=(5,5))
    # show MR image
    plt.imshow(img, cmap='gray')
    plt.imshow(mask, alpha=0.3)
    plt.axis('off')

    #plt.pause(0.01)
#%%
# assign test data location
test_dir = os.path.join('lv_deeplearning','test')
fimgs = glob.glob(os.path.join(test_dir, '*.dcm'))
gt = glob.glob(os.path.join(test_dir,'GT', '*.png'))
#%%
# create empty evaluation arrays for 20 test data
prec = np.zeros(20)
recall = np.zeros(20)
dice = np.zeros(20)
jacc = np.zeros(20)
#%%
i = 0
# Iterate over test image and predict segmentation label and calculate
evaluation metrics
for fim in fimgs:
    ds = pydicom.read_file(fim)
    img = ds.pixel_array
    mask = imread(gt[i])


    img = im_normalize(img)
    img, _ = crop_center(img, img, SZ, SZ)
    mask, _ = crop_center(mask, mask, SZ, SZ)

    test_img = img[np.newaxis, :, :, np.newaxis]
    lbl = model.predict(test_img, verbose=0).round().astype('bool')
    mask = mask.round().astype('bool')
    # Comment this line when computing approximate runtime
    display_im_label(img, lbl[0,:,:,0])
    if i == 3 or i == 12 or i == 18:
        display_im_label(img, mask)

    # calculate each evaluation metric
    # Comment this line when computing approximate runtime
    prec[i] = precision_score_(mask,lbl[0,:,:,0])
    recall[i] = recall_score_(mask,lbl[0,:,:,0])
    dice[i] = dice_coeff(mask,lbl[0,:,:,0])
    jacc[i] = iou(mask,lbl[0,:,:,0])

    i += 1

#%%
#print mean and std for each evaluation metric
print("precision mean and std:", np.mean(prec),", ", np.std(prec))
print("recall mean and std:", np.mean(recall),", ", np.std(recall))
print("dice mean and std:", np.mean(dice),", ", np.std(dice))
print("jaccard index mean and std:", np.mean(jacc),", ", np.std(jacc))
#%%
#for finding best, worst and 50th percentile samples
print(np.sort(dice))
#%%
```