Image Segmentation for Instrument Identification

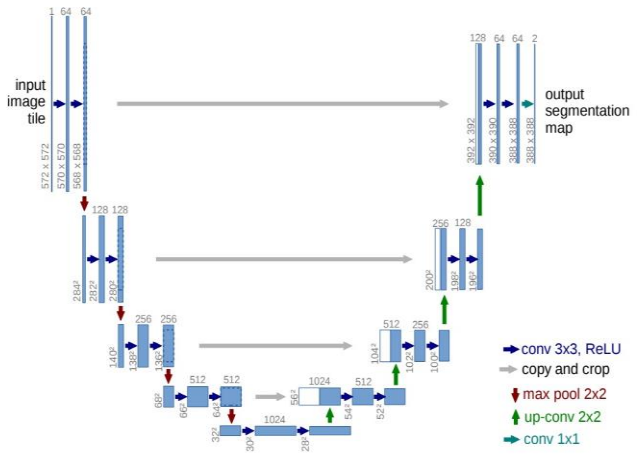
U-Net is a deep learning architecture that has been widely used in medical image segmentation. The architecture is named after its U-shape design, which consists of an encoder network that gradually reduces the spatial resolution of the input image, followed by a decoder network that gradually recovers the resolution to generate the segmentation mask.

The U-Net architecture is particularly well-suited for medical image segmentation tasks, as it can effectively capture the complex spatial relationships between different anatomical structures.

The encoder network consists of several convolutional layers that learn to extract features from the input image, while the decoder network consists of several up-sampling layers that learn to reconstruct the segmentation mask.

One of the key features of the U-Net architecture is the use of skip connections between the encoder and decoder networks. These connections allow the decoder to access information from the earlier stages of the encoder, which can help preserve fine-grained details in the segmentation mask.

The U-Net architecture has been applied to a wide range of medical image segmentation tasks, including brain tumor segmentation, lung segmentation, and cardiac segmentation, among others.



Work Done:

1. Gather the relevant dataset: Obtain the appropriate dataset for training the model.
2. Perform image masking: Apply image masking techniques to preprocess the training images.

3. Split data for training and testing: Divide the dataset into training and testing sets to evaluate model performance.
4. Experiment with different models: Test and compare different models to find the best fit for the given problem.
5. Implement the UNET algorithm: Train the model using the UNET algorithm to achieve higher accuracy.
6. Refine the model: Continuously adjust and optimize the model to enhance its performance.
7. Evaluate model performance: Test the trained model on random images and evaluate its performance based on various metrics.

Algorithm:

i. Read the original image from the training dataset and resize it into (128x128 pixels) and store it into an array X_train.

ii. Read the mask of training images as grayscale and binarize it, resize into (128x128 pixels) and store it into array Y_train.

iii. Read and resize the images in test folder and store it into array X_test.

iv. Creates an input layer for the model using the Input class from the **tf.keras.layers** module. The layer is configured to accept inputs with the shape (IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)

v. Create a normalization layer for the model using the Lambda class from the **tf.keras.layers** module. The layer applies a lambda function that divides each pixel value by 255 to normalize the input images. The layer takes the inputs layer as input and returns the normalized output as the variable s. The lambda function is applied element-wise to the input tensor using TensorFlow's automatic broadcasting.

vi. **c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)**This line defines the first convolutional layer of the U-Net encoder. The layer takes the input tensor s as input and applies 16 filters with a kernel size of 3x3. The activation argument sets the activation function to **ReLU**, and **padding='same'** sets the padding mode to keep the output shape the same as the input shape. The kernel_initializer argument sets the weight initialization method to He normal initialization.

vii. **c1 = tf.keras.layers.Dropout(0.1)(c1):** This line adds a dropout layer to the model, which randomly sets a fraction of the input units to zero during training to prevent overfitting. The 0.1 argument sets the dropout rate to 10%

viii. **model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])**

ix. The optimizer parameter specifies the optimization algorithm to use during training, and in this case, it is set to 'adam'. Adam is a popular optimization algorithm for deep learning models that adaptively adjusts the learning rate and other parameters during training.

x. The loss parameter specifies the loss function to use during training, and in this case, it is set to 'binary_crossentropy'. Binary cross-entropy is a commonly used loss function for binary classification problems, where the goal is to predict a binary outcome (e.g., true/false, 0/1).

xi. The metrics parameter specifies the evaluation metrics to compute during training and testing, and in this case, it is set to ['accuracy']. This means that the model's accuracy will be computed and reported after each epoch (i.e., complete pass through the training data) during training, and also after testing on the validation data.

xii. Overall, this line of code sets up the model for training with the Adam optimizer, using binary cross-entropy loss, and monitoring accuracy as the evaluation metric

xiii. **checkpointer = tf.keras.callbacks.ModelCheckpoint('model_instrument_seg.h5', verbose=1, save_best_only=True)**

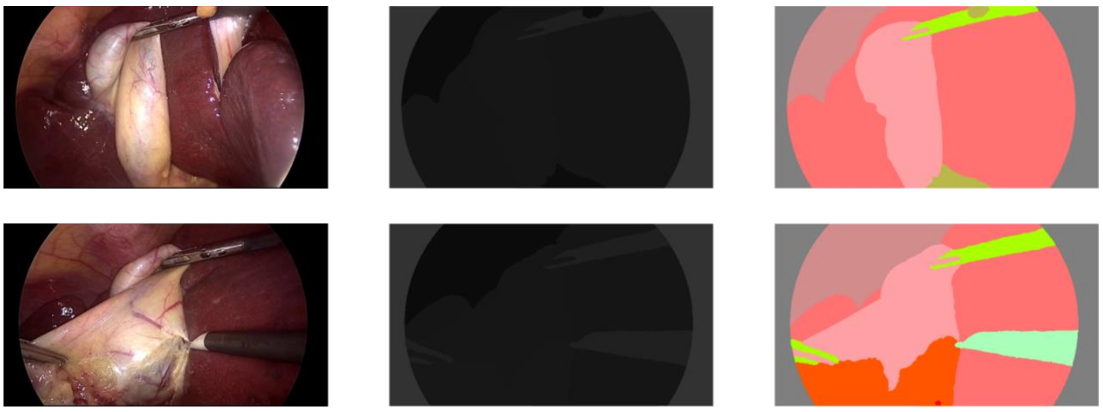
xiv. This line of code is creating an instance of a Keras callback called ModelCheckpoint. The ModelCheckpoint callback is used to save the model weights at certain intervals during training.

xv. save_best_only=True: a boolean flag indicating whether to save only the weights that achieve the best performance on the validation set. If set to True, the callback will only save the weights if the model's performance on the validation set has improved since the last save. This can help prevent overfitting by avoiding the selection of weights that may have overfit to the training data.

xvi. fitting a machine learning model using the training data X_train and Y_train, with a validation split of 0.1 (meaning 10% of the training data will be used for validation). It uses a batch size of 16 and trains for 25 epochs. The callbacks argument specifies a list of callbacks to apply during training, which in this case includes the checkpointer callback to save the best model during training. The results of the training will be returned by the fit method.

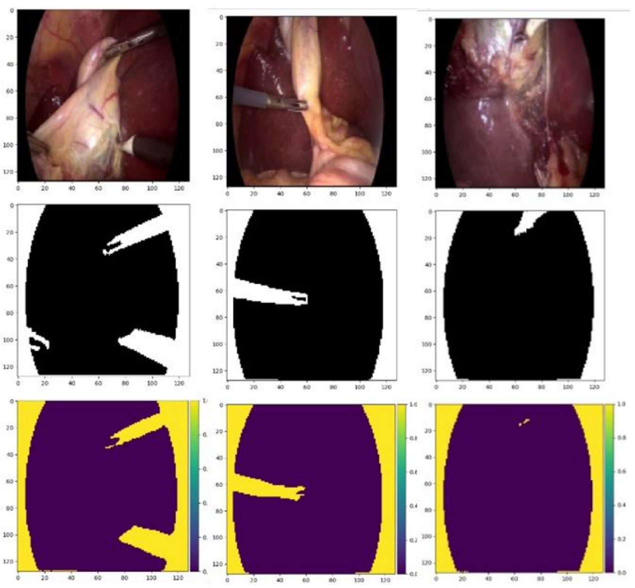
xvii. It can be observed that the accuracy of the model is very high in the range of (89% – 97%) and it performs better segmentation.

Sample Dataset



Test Results:

After training the model, it was subjected to a testing phase using a set of random images. During this evaluation, it was observed that the model was able to accurately identify and differentiate the instruments from the organs, effectively performing the task of image segmentation



Submitted by: Hitika Dalwadi

Submitted to: Dr. Kunal Korgaonkar