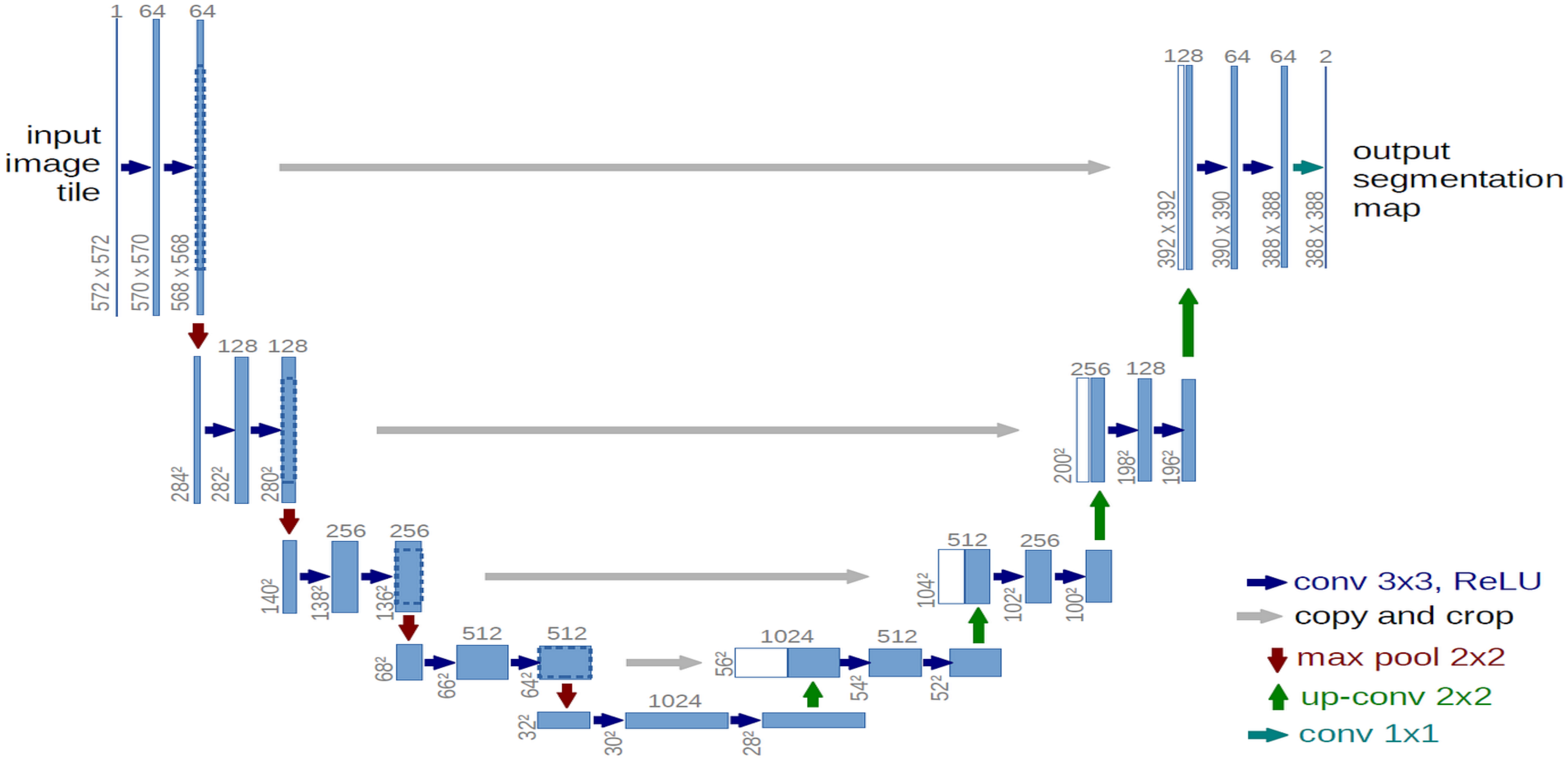
Binary Segmentation of Medical Surgical Tool

Consent By:-
Rameshwar Singh

Consent To :-
Er. Urooj Khan
Data Science Trainer

Contents

• Unet Architecture



- U-Net is a convolutional neural network (CNN) architecture that was developed in 2015 for biomedical image segmentation.
- It is a U-shaped encoder-decoder network architecture that consists of four encoder blocks and four decoder blocks.
- The encoder network (contracting path) halves the spatial dimensions and doubles the number of filters at each encoder block.

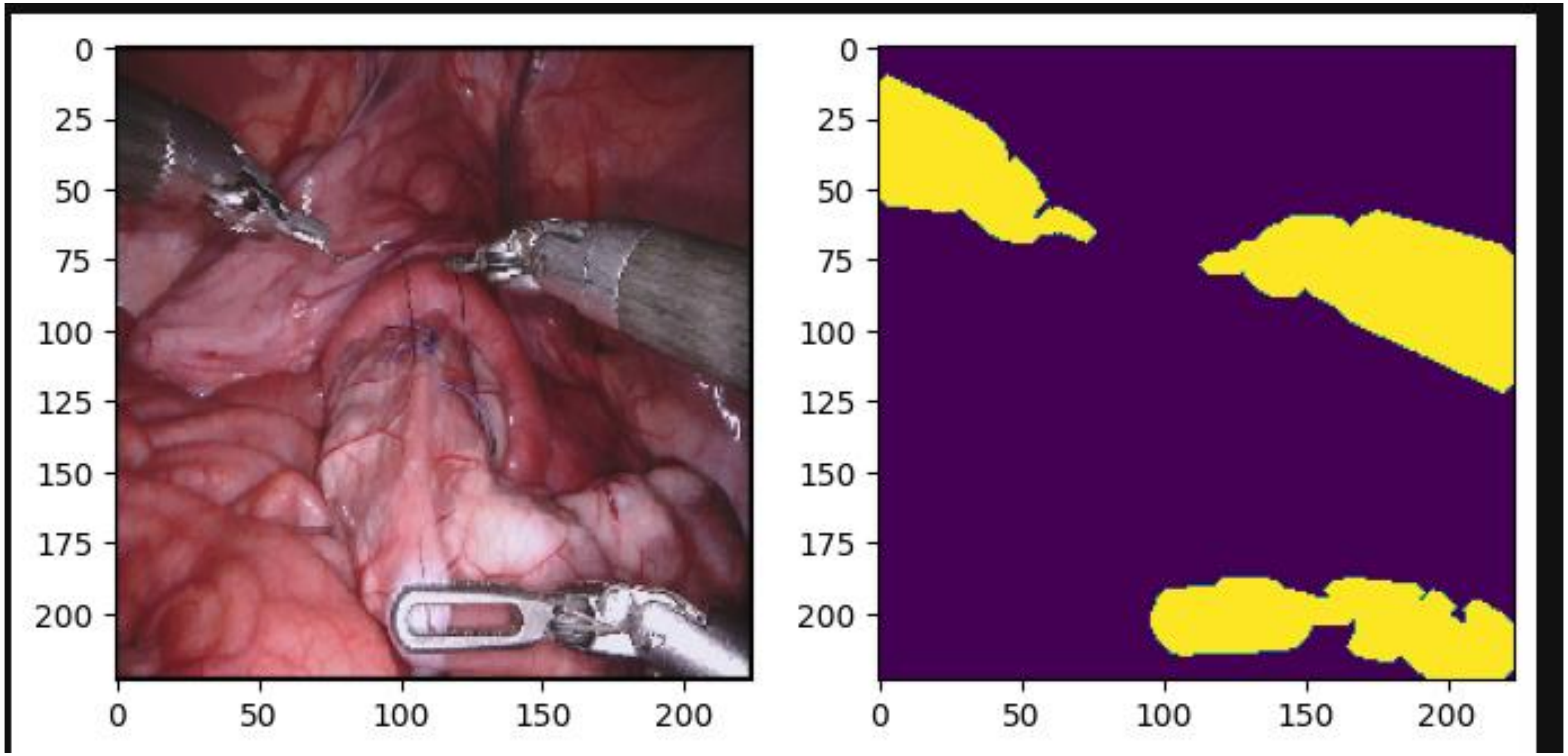
Preprocessing.

Preprocessing Include application of Image Processing Techniques including

Image Acquisition, Image Enhancement, Image Filtering, Geometric Transformations, Colour Processing, Image Restoration, Morphological Operations, Feature Extraction over the Dataset.

DATASET.

I have used 512 x 512 size of imaging tiles of Surgical Tool. With their Respected binary segmented map



model

```
def conv_block(tensor, nfilters, size=3, padding='same', initializer="he_normal"):
  x = Conv2D(filters=nfilters, kernel_size=(size, size), padding=padding,   kernel_initializer=initializer)(tensor)
   x = BatchNormalization()(x)
   x = Activation("relu")(x)
   x = Conv2D(filters=nfilters, kernel_size=(size, size), padding=padding, kernel_initializer=initializer)(x)
   x = BatchNormalization()(x)
   x = Activation("relu")(x)
   return x

def deconv_block(tensor, residual, nfilters, size=3, padding='same', strides=(2, 2)):
   y = Conv2DTranspose(nfilters, kernel_size=(size, size), strides=strides, padding=padding)(tensor)
   y = concatenate([y, residual], axis=3)
   y = conv_block(y, nfilters)
   return y
```

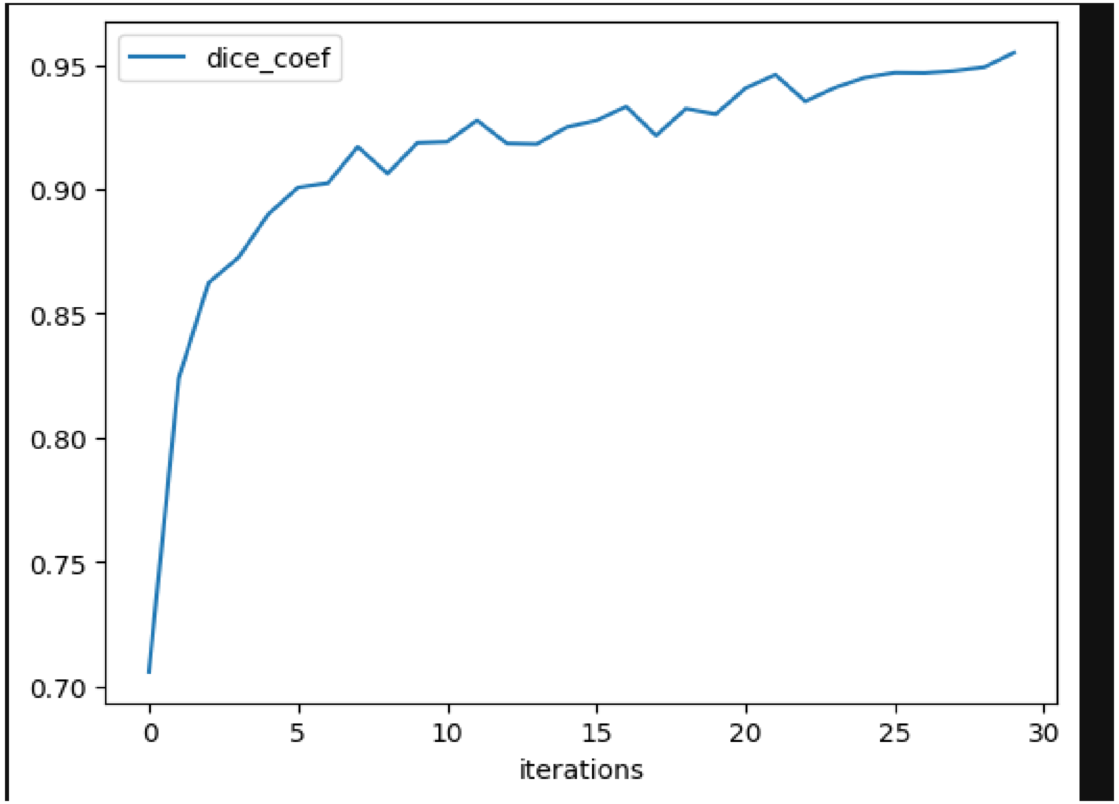
```
def Unet(h, w, filters):

# down
    input_layer = Input(shape=(h, w, 3), name='image_input')
    conv1 = conv_block(input_layer, nfilters=filters)
    conv1_out = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = conv_block(conv1_out, nfilters=filters*2)
    conv2_out = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = conv_block(conv2_out, nfilters=filters*4)
    conv3_out = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv4 = conv_block(conv3_out, nfilters=filters*8)
    conv4_out = MaxPooling2D(pool_size=(2, 2))(conv4)
    conv4_out = Dropout(0.5)(conv4_out)
    conv5 = conv_block(conv4_out, nfilters=filters*16)
    conv5 = Dropout(0.5)(conv5)
```

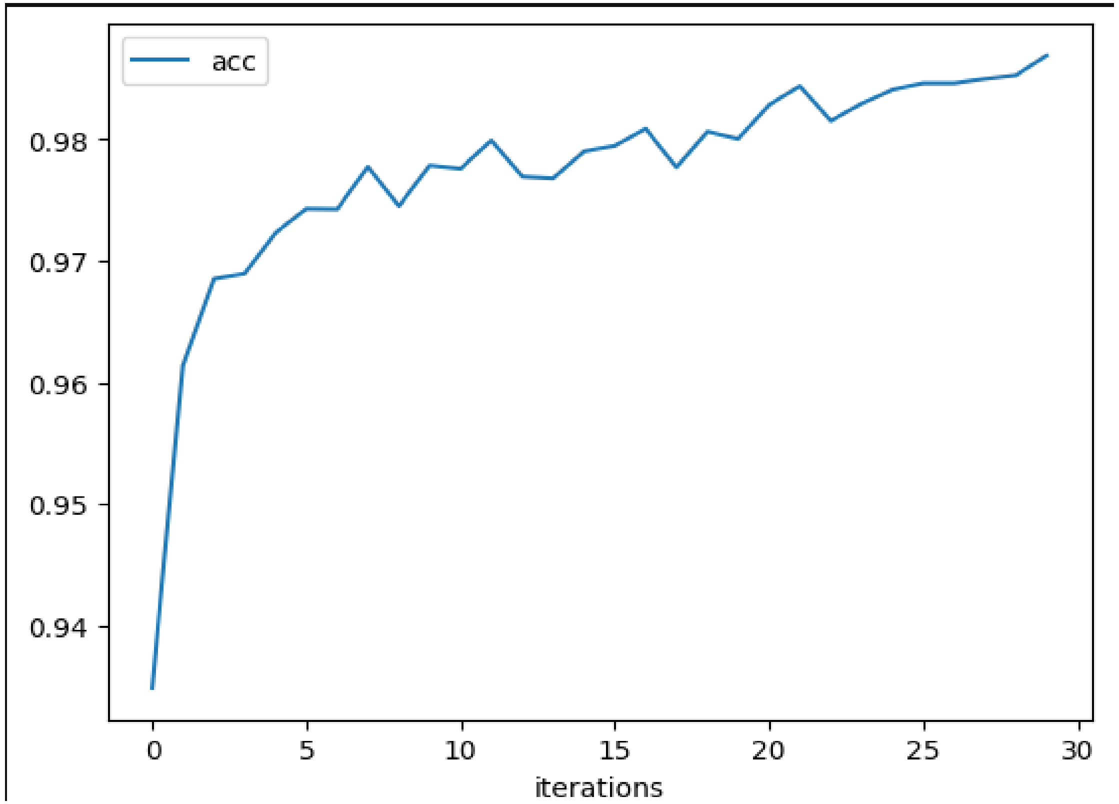
```
deconv6 = deconv_block(conv5, residual=conv4, nfilters=filters*8)
 deconv6 = Dropout(0.5)(deconv6)
 deconv7 = deconv_block(deconv6, residual=conv3, nfilters=filters*4)
 deconv7 = Dropout(0.5)(deconv7)
 deconv8 = deconv_block(deconv7, residual=conv2, nfilters=filters*2)
 deconv9 = deconv_block(deconv8, residual=conv1, nfilters=filters)
 output_layer = Conv2D(filters=1, kernel_size=(1, 1), activation='sigmoid')(deconv9)
 # using sigmoid activation for binary classification
 model = Model(inputs=input_layer, outputs=output_layer, name='Unet')
 return model
```

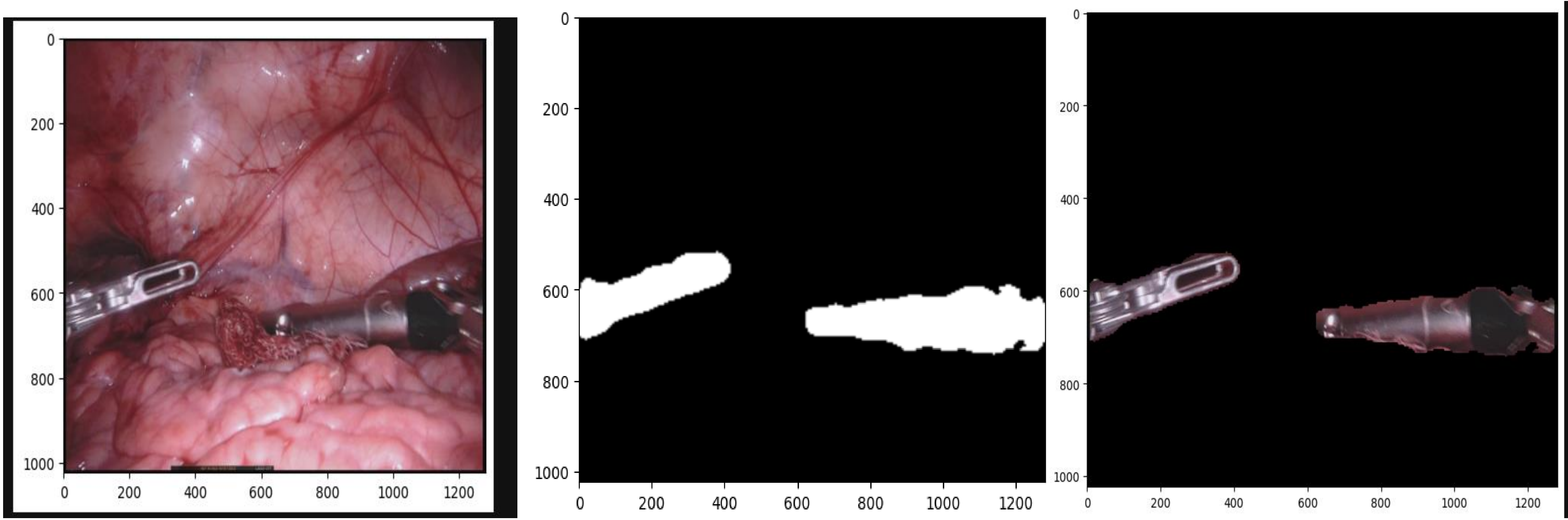
- def jaccard_distance_loss(y_true, y_pred,smooth = 100):   intersection = K.sum(K.abs(y_true * y_pred), axis=-1)   sum_ = K.sum(K.abs(y_true) + K.abs(y_pred), axis=-1)   jac = (intersection + smooth) / (sum_ - intersection + smooth)   return (1 - jac) * smooth

- 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 + K.epsilon()) / (K.sum(y_true_f) + K.sum(y_pred_f) + K.epsilon())

Dice_coef & Accuracy






0
200
400
600
800
1000
0
200
400
600
800
1000
1200
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
0
200
400
600
800
1000
0
200
400
600
800
1000
1200

THANK YOU