Image_Colorization_Generic_GAN_Model

March 24, 2023

1 Image Colorization with GANs

[ ]:
```
from PIL import Image
from sklearn.model_selection import train_test_split
import tensorflow as tf
import numpy as np
from matplotlib import image
from matplotlib import pyplot as plt
import os
from tensorflow import keras
```

1.1 Data load

[ ]:
```
batch_size = 64
img_size = 120
dataset_split = 2500

master_dir = 'Endoscopy/images/'
x = []
y = []
for image_file in os.listdir( master_dir )[ 0 : dataset_split ]:
    rgb_image = Image.open( os.path.join( master_dir , image_file ) ).resize( (
 ↪img_size , img_size ) )
    rgb_img_array = (np.asarray( rgb_image ) ) / 255
    gray_image = rgb_image.convert( 'L' )
    gray_img_array = ( np.asarray( gray_image ).reshape( ( img_size , img_size
 ↪, 1 ) ) ) / 255
    x.append( gray_img_array )
    y.append( rgb_img_array )

# Train-test splitting
train_x, test_x, train_y, test_y = train_test_split( np.array(x) , np.array(y)
 ↪, test_size=0.1 )

# Construct tf.data.Dataset object
dataset = tf.data.Dataset.from_tensor_slices( ( train_x , train_y ) )
dataset = dataset.batch( batch_size )
```

1.1.1 GAN: Generator

[3]:
```
def get_generator_model():

    inputs = tf.keras.layers.Input( shape=( img_size , img_size , 1 ) )

    conv1 = tf.keras.layers.Conv2D( 16 , kernel_size=( 5 , 5 ) , strides=1 )(␣
↪inputs )
    conv1 = tf.keras.layers.LeakyReLU()( conv1 )
    conv1 = tf.keras.layers.Conv2D( 32 , kernel_size=( 3 , 3 ) , strides=1)(␣
↪conv1 )
    conv1 = tf.keras.layers.LeakyReLU()( conv1 )
    conv1 = tf.keras.layers.Conv2D( 32 , kernel_size=( 3 , 3 ) , strides=1)(␣
↪conv1 )
    conv1 = tf.keras.layers.LeakyReLU()( conv1 )

    conv2 = tf.keras.layers.Conv2D( 32 , kernel_size=( 5 , 5 ) , strides=1)(␣
↪conv1 )
    conv2 = tf.keras.layers.LeakyReLU()( conv2 )
    conv2 = tf.keras.layers.Conv2D( 64 , kernel_size=( 3 , 3 ) , strides=1 )(␣
↪conv2 )
    conv2 = tf.keras.layers.LeakyReLU()( conv2 )
    conv2 = tf.keras.layers.Conv2D( 64 , kernel_size=( 3 , 3 ) , strides=1 )(␣
↪conv2 )
    conv2 = tf.keras.layers.LeakyReLU()( conv2 )

    conv3 = tf.keras.layers.Conv2D( 64 , kernel_size=( 5 , 5 ) , strides=1 )(␣
↪conv2 )
    conv3 = tf.keras.layers.LeakyReLU()( conv3 )
    conv3 = tf.keras.layers.Conv2D( 128 , kernel_size=( 3 , 3 ) , strides=1 )(␣
↪conv3 )
    conv3 = tf.keras.layers.LeakyReLU()( conv3 )
    conv3 = tf.keras.layers.Conv2D( 128 , kernel_size=( 3 , 3 ) , strides=1 )(␣
↪conv3 )
    conv3 = tf.keras.layers.LeakyReLU()( conv3 )

    bottleneck = tf.keras.layers.Conv2D( 128 , kernel_size=( 3 , 3 ) ,␣
↪strides=1 , activation='tanh' , padding='same' )( conv3 )

    concat_1 = tf.keras.layers.Concatenate()( [ bottleneck , conv3 ] )
    conv_up_3 = tf.keras.layers.Conv2DTranspose( 128 , kernel_size=( 3 , 3 ) ,␣
↪strides=1 , activation='relu' )( concat_1 )
    conv_up_3 = tf.keras.layers.Conv2DTranspose( 128 , kernel_size=( 3 , 3 ) ,␣
↪strides=1 , activation='relu' )( conv_up_3 )
    conv_up_3 = tf.keras.layers.Conv2DTranspose( 64 , kernel_size=( 5 , 5 ) ,␣
↪strides=1 , activation='relu' )( conv_up_3 )
```

```
    concat_2 = tf.keras.layers.Concatenate()( [ conv_up_3 , conv2 ] )
    conv_up_2 = tf.keras.layers.Conv2DTranspose( 64 , kernel_size=( 3 , 3 ) ,␣
↪strides=1 , activation='relu' )( concat_2 )
    conv_up_2 = tf.keras.layers.Conv2DTranspose( 64 , kernel_size=( 3 , 3 ) ,␣
↪strides=1 , activation='relu' )( conv_up_2 )
    conv_up_2 = tf.keras.layers.Conv2DTranspose( 32 , kernel_size=( 5 , 5 ) ,␣
↪strides=1 , activation='relu' )( conv_up_2 )

    concat_3 = tf.keras.layers.Concatenate()( [ conv_up_2 , conv1 ] )
    conv_up_1 = tf.keras.layers.Conv2DTranspose( 32 , kernel_size=( 3 , 3 ) ,␣
↪strides=1 , activation='relu')( concat_3 )
    conv_up_1 = tf.keras.layers.Conv2DTranspose( 32 , kernel_size=( 3 , 3 ) ,␣
↪strides=1 , activation='relu')( conv_up_1 )
    conv_up_1 = tf.keras.layers.Conv2DTranspose( 3 , kernel_size=( 5 , 5 ) ,␣
↪strides=1 , activation='relu')( conv_up_1 )

    model = tf.keras.models.Model( inputs , conv_up_1 )
    return model
```

1.1.2 GAN:Discriminator

[4]:
```
def get_discriminator_model():
    layers = [
        tf.keras.layers.Conv2D( 32 , kernel_size=( 7 , 7 ) , strides=1 ,␣
↪activation='relu' , input_shape=( 120 , 120 , 3 ) ),
        tf.keras.layers.Conv2D( 32 , kernel_size=( 7, 7 ) , strides=1,␣
↪activation='relu'  ),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D( 64 , kernel_size=( 5 , 5 ) , strides=1,␣
↪activation='relu'  ),
        tf.keras.layers.Conv2D( 64 , kernel_size=( 5 , 5 ) , strides=1,␣
↪activation='relu'  ),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D( 128 , kernel_size=( 3 , 3 ) , strides=1,␣
↪activation='relu'  ),
        tf.keras.layers.Conv2D( 128 , kernel_size=( 3 , 3 ) , strides=1,␣
↪activation='relu'  ),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D( 256 , kernel_size=( 3 , 3 ) , strides=1,␣
↪activation='relu'  ),
        tf.keras.layers.Conv2D( 256 , kernel_size=( 3 , 3 ) , strides=1,␣
↪activation='relu'  ),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense( 512, activation='relu'  )  ,
        tf.keras.layers.Dense( 128 , activation='relu' ) ,
```

```
        tf.keras.layers.Dense( 16 , activation='relu' ) ,
        tf.keras.layers.Dense( 1 , activation='sigmoid' )
    ]
    model = tf.keras.models.Sequential( layers )
    return model
```

1.1.3 Loss Functions

[5]:
```
cross_entropy = tf.keras.losses.BinaryCrossentropy()
mse = tf.keras.losses.MeanSquaredError()

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output) - tf.random.uniform(␣
 ↪shape=real_output.shape , maxval=0.1 ) , real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output) + tf.random.uniform(␣
 ↪shape=fake_output.shape , maxval=0.1  ) , fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output , real_y):
    real_y = tf.cast( real_y , 'float32' )
    return mse( fake_output , real_y )

generator_optimizer = tf.keras.optimizers.Adam( 0.0005 )
discriminator_optimizer = tf.keras.optimizers.Adam( 0.0005 )

generator = get_generator_model()
discriminator = get_discriminator_model()
```

1.2 Training GAN

[6]:
```
@tf.function
def train_step( input_x , real_y ):

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator( input_x , training=True)
        real_output = discriminator( real_y, training=True)
        generated_output = discriminator(generated_images, training=True)

        # L2 Loss
        gen_loss = generator_loss( generated_images , real_y )
        disc_loss = discriminator_loss( real_output, generated_output )

    # Compute the gradients
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.
 ↪trainable_variables)
```

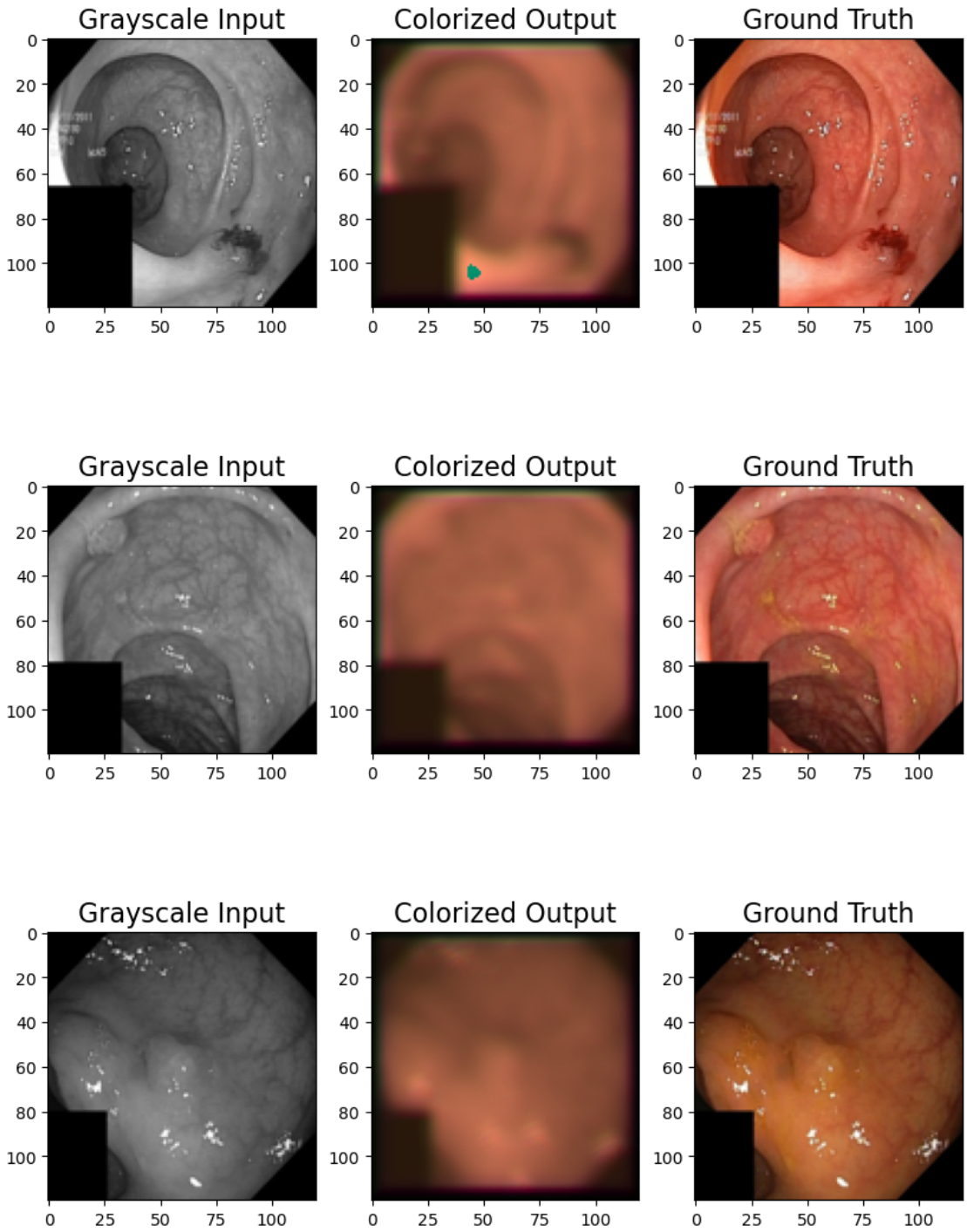
```
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.
↪trainable_variables)

    # Optimize with Adam
    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.
↪trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,␣
↪discriminator.trainable_variables))
```

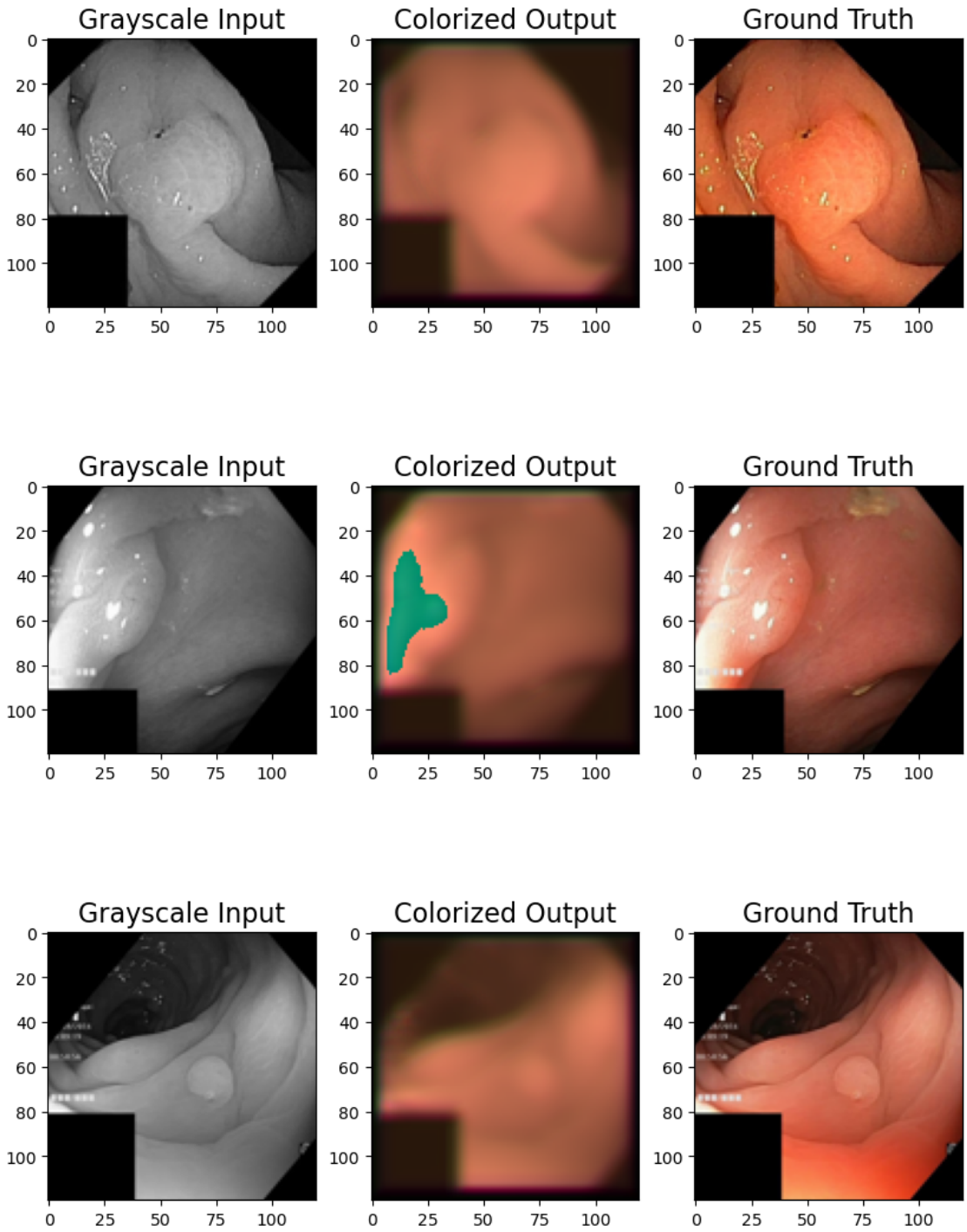
[12]:
```
num_epochs = 10

for e in range( num_epochs ):
    print( e )
    for ( x , y ) in dataset:
        print( '<==========done=========>')
        train_step( x , y )
```

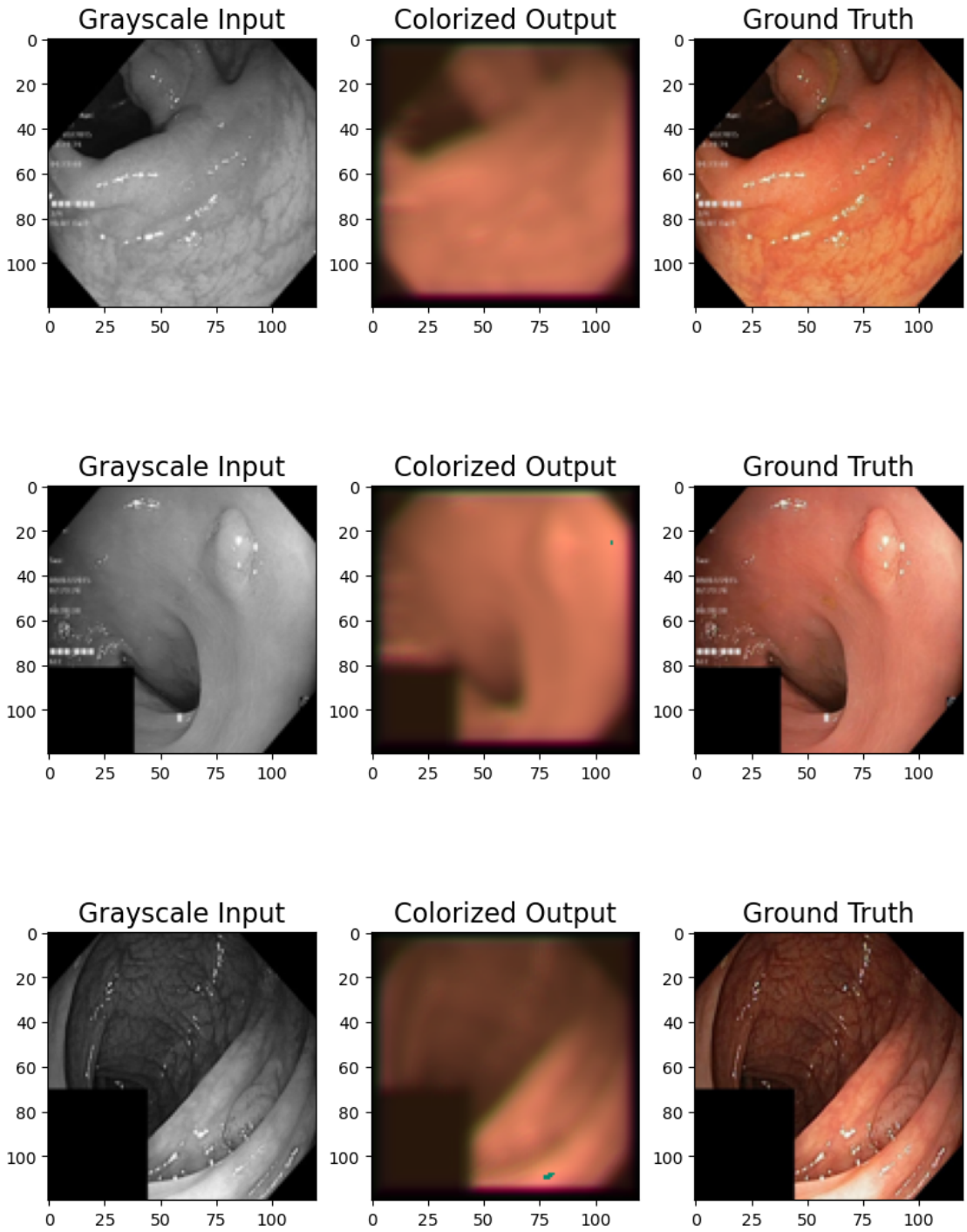
1.3 Results

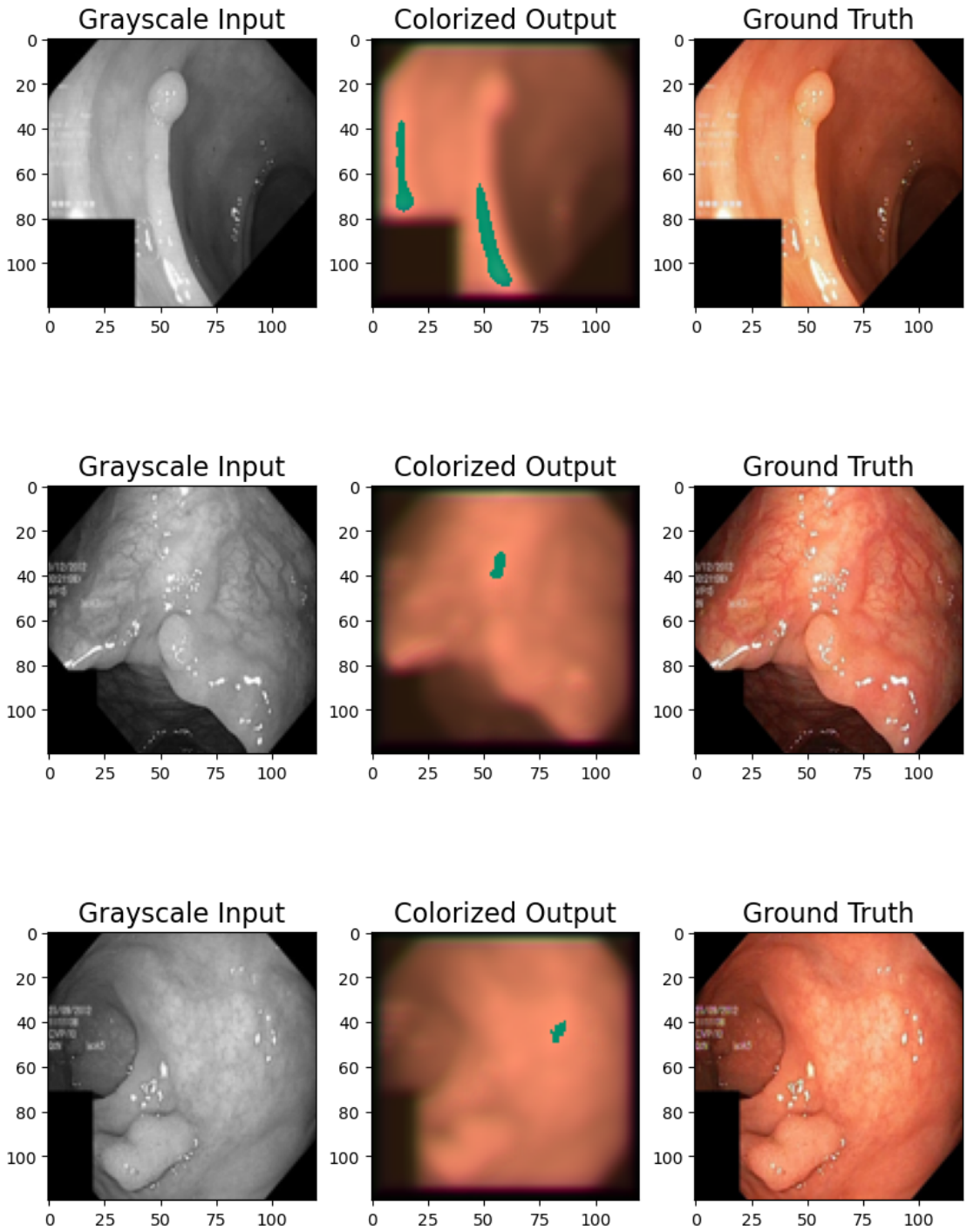
[17]:
```
y = generator( test_x[0 : ] ).numpy()
```

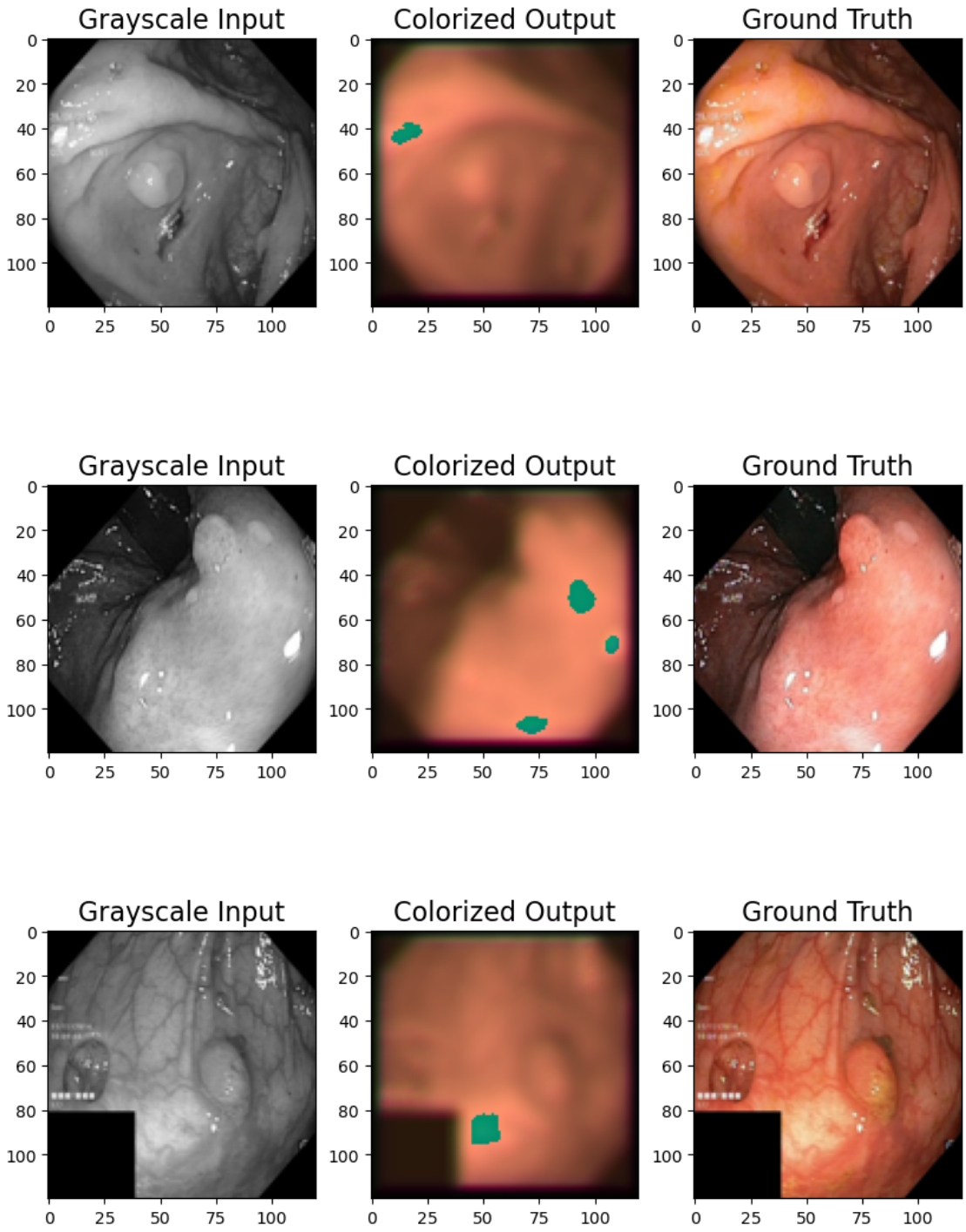
[18]:
```
H,W=[120,120]
for i in range(len(test_x)):
    plt.figure(figsize=(10,10))
    or_image = plt.subplot(3,3,1)
    or_image.set_title('Grayscale Input', fontsize=16)
    plt.imshow( test_x[i].reshape((H,W)) , cmap='gray' )

    in_image = plt.subplot(3,3,2)
    image = Image.fromarray( ( y[i] * 255 ).astype( 'uint8' ) ).resize( ( H , W␣
↪) )
    image = np.asarray( image )
    in_image.set_title('Colorized Output', fontsize=16)
    plt.imshow( image )

    ou_image = plt.subplot(3,3,3)
    image = Image.fromarray( ( test_y[i] * 255 ).astype( 'uint8' ) ).resize( (␣
↪H , W ) )
    ou_image.set_title('Ground Truth', fontsize=16)
    plt.imshow( image )


    plt.show()
```

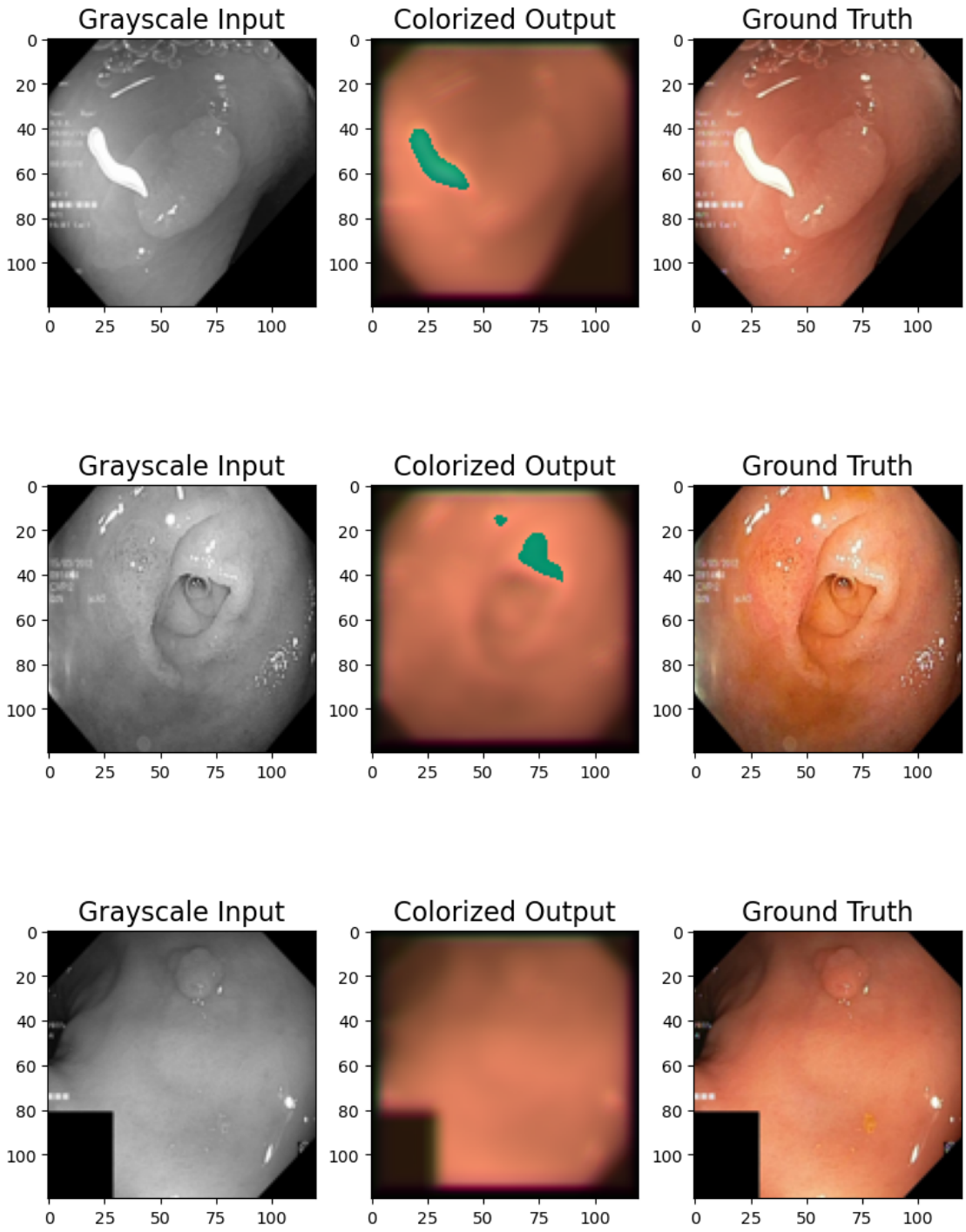

Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth

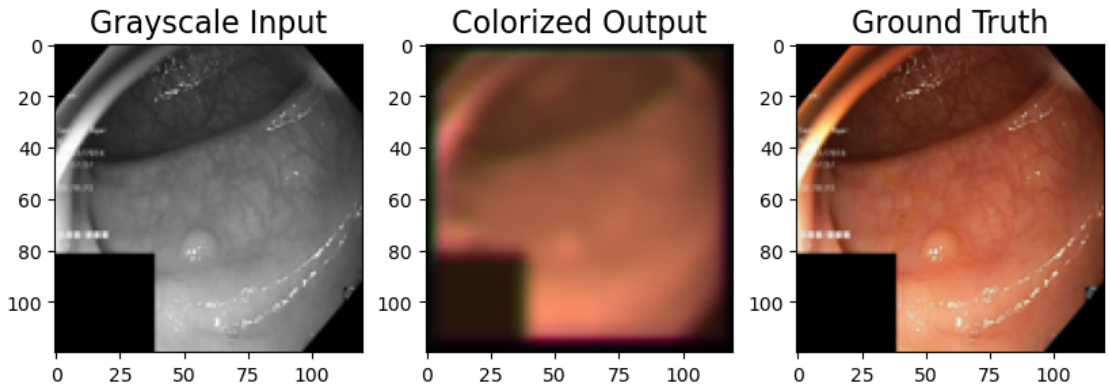

Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth

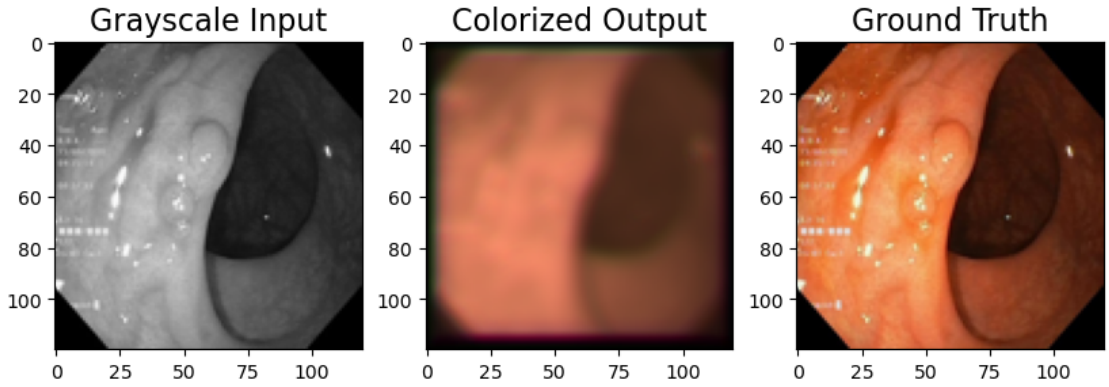

Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth


Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth


Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth


Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth
Grayscale Input
Colorized Output
Ground Truth


Grayscale Input
Colorized Output
Ground Truth
0
20
40
60
80
100
0
25
50
75
100


Grayscale Input
Colorized Output
Ground Truth
0
20
40
60
80
100
0
25
50
75
100

[ ]:

[ ]: