```
!rm -rf clone && git clone https://github.com/Bileth/Grayscale-image-colorization clone &&
```

```
Cloning into 'clone'...
remote: Enumerating objects: 14317, done.
remote: Counting objects: 100% (14317/14317), done.
remote: Compressing objects: 100% (14311/14311), done.
remote: Total 14317 (delta 5), reused 14304 (delta 1), pack-reused 0
Receiving objects: 100% (14317/14317), 230.02 MiB | 21.16 MiB/s, done.
Resolving deltas: 100% (5/5), done.
Checking out files: 100% (22296/22296), done.
```

```
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

# Veličina serije koju ćemo koristiti za obuku
batch_size = 64

# Size of the image required to train our model
img_size = 120

# Veličina slika potrebnih za treniranje našeg modela
dataset_split = 2500

master_dir = 'train/images/color/'
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 ,
    # Normalize the RGB image array
    rgb_img_array = (np.asarray( rgb_image ) ) / 255
    gray_image = rgb_image.convert( 'L' )
    # Normaliziranje RGB niza slika
    gray_img_array = ( np.asarray( gray_image ).reshape( ( img_size , img_size , 1 ) ) ) /
    # Nadodavnje oba niza slika
    x.append( gray_img_array )
    y.append( rgb_img_array )

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

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

```
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 , activatio

    concat_1 = tf.keras.layers.Concatenate()( [ bottleneck , conv3 ] )
    conv_up_3 = tf.keras.layers.Conv2DTranspose( 128 , kernel_size=( 3 , 3 ) , strides=1 , a
    conv_up_3 = tf.keras.layers.Conv2DTranspose( 128 , kernel_size=( 3 , 3 ) , strides=1 , a
    conv_up_3 = tf.keras.layers.Conv2DTranspose( 64 , kernel_size=( 5 , 5 ) , strides=1 , ac

    concat_2 = tf.keras.layers.Concatenate()( [ conv_up_3 , conv2 ] )
    conv_up_2 = tf.keras.layers.Conv2DTranspose( 64 , kernel_size=( 3 , 3 ) , strides=1 , ac
    conv_up_2 = tf.keras.layers.Conv2DTranspose( 64 , kernel_size=( 3 , 3 ) , strides=1 , ac
    conv_up_2 = tf.keras.layers.Conv2DTranspose( 32 , kernel_size=( 5 , 5 ) , strides=1 , ac

    concat_3 = tf.keras.layers.Concatenate()( [ conv_up_2 , conv1 ] )
    conv_up_1 = tf.keras.layers.Conv2DTranspose( 32 , kernel_size=( 3 , 3 ) , strides=1 , ac
    conv_up_1 = tf.keras.layers.Conv2DTranspose( 32 , kernel_size=( 3 , 3 ) , strides=1 , ac
    conv_up_1 = tf.keras.layers.Conv2DTranspose( 3 , kernel_size=( 5 , 5 ) , strides=1 , act

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


def get_discriminator_model():
    layers = [
              tf.keras.layers.Conv2D( 32 , kernel_size=( 7 , 7 ) , strides=1 , activation='r
              tf.keras.layers.Conv2D( 32 , kernel_size=( 7, 7 ) , strides=1, activation='rel
              tf.keras.layers.MaxPooling2D(),
              tf.keras.layers.Conv2D( 64 , kernel_size=( 5 , 5 ) , strides=1, activation='re
              tf.keras.layers.Conv2D( 64 , kernel_size=( 5 , 5 ) , strides=1, activation='re
              tf.keras.layers.MaxPooling2D(),
              tf.keras.layers.Conv2D( 128 , kernel_size=( 3 , 3 ) , strides=1, activation='r
              tf.keras.layers.Conv2D( 128 , kernel_size=( 3 , 3 ) , strides=1, activation='r
              tf.keras.layers.MaxPooling2D(),
```

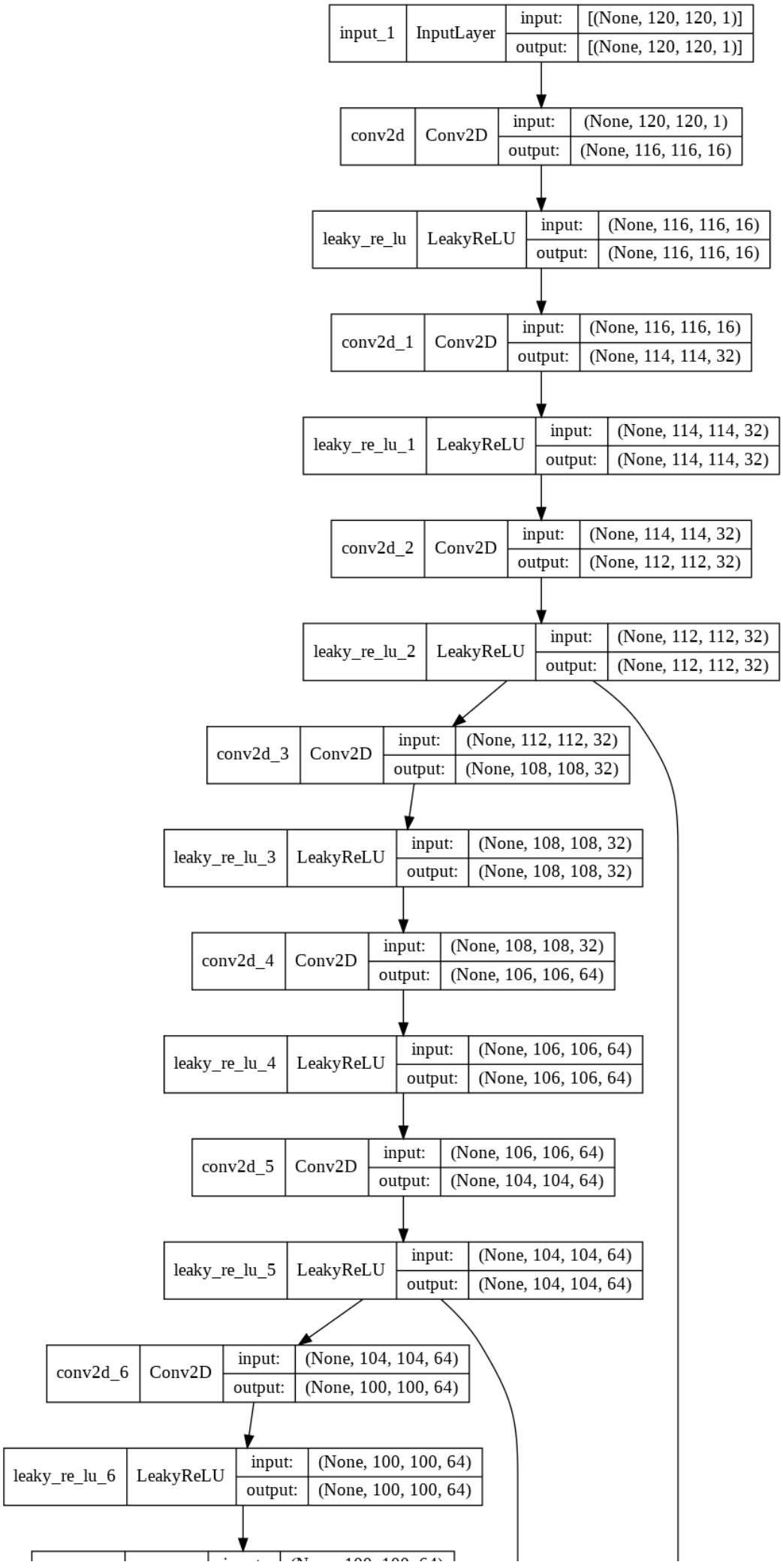
```
            tf.keras.layers.Conv2D( 256 , kernel_size=( 3 , 3 ) , strides=1, activation='r
            tf.keras.layers.Conv2D( 256 , kernel_size=( 3 , 3 ) , strides=1, activation='r
            tf.keras.layers.MaxPooling2D(),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense( 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

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_ou
    fake_loss = cross_entropy(tf.zeros_like(fake_output) + tf.random.uniform( shape=fake_o
    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()


@tf.function
def train_step( input_x , real_y ):

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        # generiranje slike -> G( x )
        generated_images = generator( input_x , training=True)
        # Vjerojatnost da je slika prava (točna) -> D( x )
        real_output = discriminator( real_y, training=True)
        # Vjerojatnost da je slika generirana  -> D( G( x ) )
        generated_output = discriminator(generated_images, training=True)

        # L2 gubitak -> || y - G(x) ||^2
        gen_loss = generator_loss( generated_images , real_y )
        # Log gubitka za diskriminator
        disc_loss = discriminator_loss( real_output, generated_output )

    #tf.keras.backend.print_tensor( tf.keras.backend.mean( gen_loss ) )
    #tf.keras.backend.print_tensor( gen_loss + disc_loss )

    # Računanje gradijenata
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_var
```

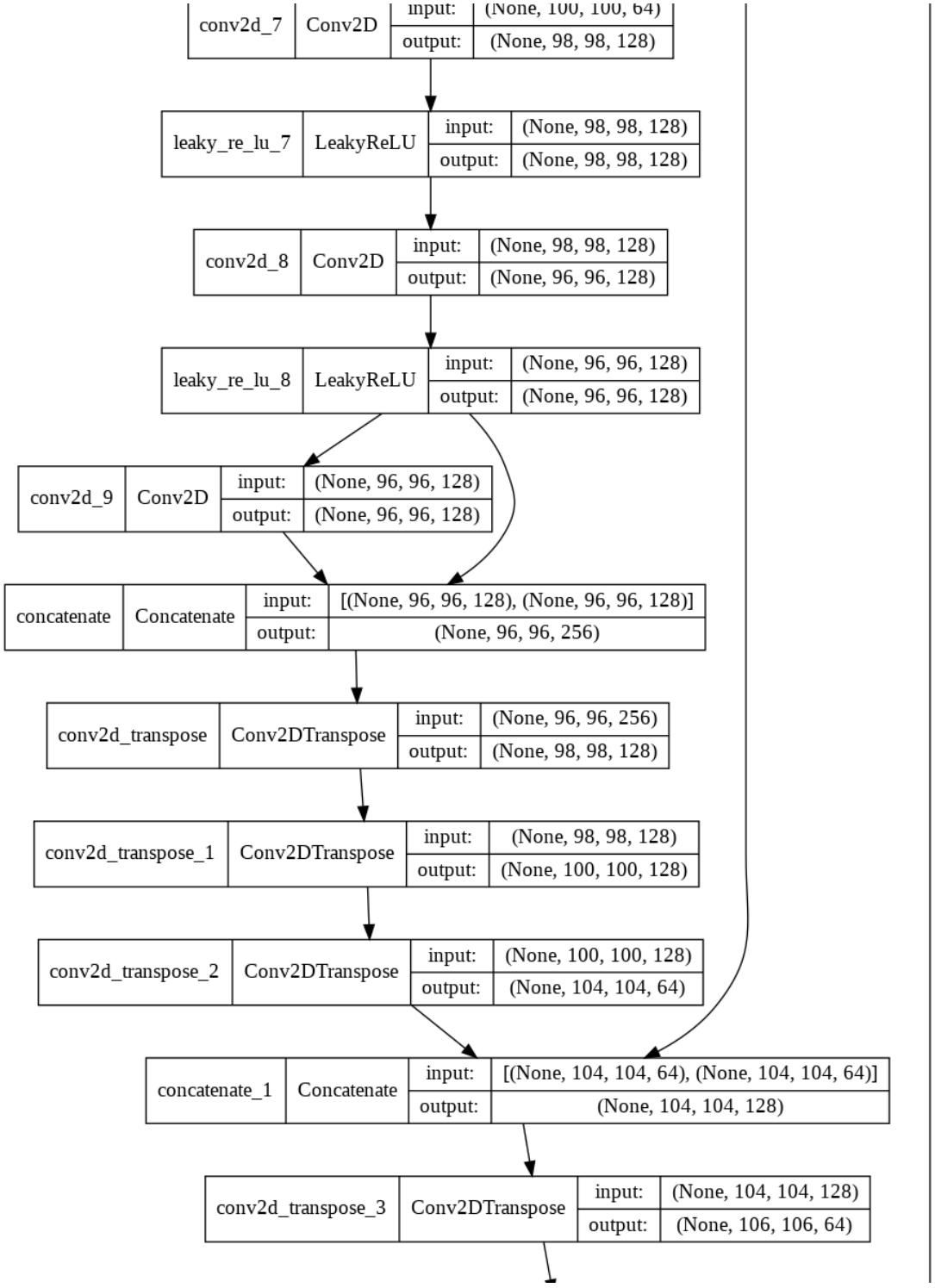
```
    # optimizacija sa Adam-om
    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_va
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.


num_epochs = 150

for e in range( num_epochs ):
    print( e )
    for ( x , y ) in dataset:
        # Ovjde ( x , y ) predstavlja skup iz našeg skupa podataka za obuku.
        print( x.shape )
        train_step( x , y )
```

```
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(10, 120, 120, 1)
148
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(10, 120, 120, 1)
149
(64, 120, 120, 1)
(64, 120, 120, 1)
```

```
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
(64, 120, 120, 1)
```

```
generator.compile(optimizer='adam', loss='binary_crossentropy')
tf.keras.utils.plot_model(generator, 'encoder_compress.png', show_shapes=True)
```


input_1 InputLayer input: [(None, 120, 120, 1)] output: [(None, 120, 120, 1)]
conv2d Conv2D input: (None, 120, 120, 1) output: (None, 116, 116, 16)
leaky_re_lu LeakyReLU input: (None, 116, 116, 16) output: (None, 116, 116, 16)
conv2d_1 Conv2D input: (None, 116, 116, 16) output: (None, 114, 114, 32)
leaky_re_lu_1 LeakyReLU input: (None, 114, 114, 32) output: (None, 114, 114, 32)
conv2d_2 Conv2D input: (None, 114, 114, 32) output: (None, 112, 112, 32)
leaky_re_lu_2 LeakyReLU input: (None, 112, 112, 32) output: (None, 112, 112, 32)
conv2d_3 Conv2D input: (None, 112, 112, 32) output: (None, 108, 108, 32)
leaky_re_lu_3 LeakyReLU input: (None, 108, 108, 32) output: (None, 108, 108, 32)
conv2d_4 Conv2D input: (None, 108, 108, 32) output: (None, 106, 106, 64)
leaky_re_lu_4 LeakyReLU input: (None, 106, 106, 64) output: (None, 106, 106, 64)
conv2d_5 Conv2D input: (None, 106, 106, 64) output: (None, 104, 104, 64)
leaky_re_lu_5 LeakyReLU input: (None, 104, 104, 64) output: (None, 104, 104, 64)
conv2d_6 Conv2D input: (None, 104, 104, 64) output: (None, 100, 100, 64)
leaky_re_lu_6 LeakyReLU input: (None, 100, 100, 64) output: (None, 100, 100, 64)




```
generator.save('train/my_model')
```

```
INFO:tensorflow:Assets written to: train/my_model/assets
```

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

```
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((120,120)) , cmap='gray' )

  in_image = plt.subplot(3,3,2)
  image = Image.fromarray( ( y[i] * 255 ).astype( 'uint8' ) ).resize( ( 1024 , 1024 ) )
```

```
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( ( 1024 , 1024 )
ou_image.set_title('Ground Truth', fontsize=16)
plt.imshow( image )

plt.savefig("plots.pdf")
plt.show()
```