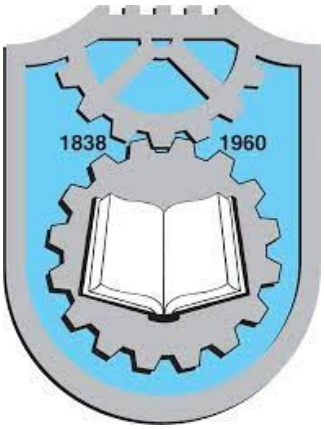
Универзитет у Крагујевцу
Факултет инжењерских наука



Тема:

Пребацивање црно-белих слика у RGB слике

Студенти:
Данило Николић 617/2019
Иван Радивојевић 619/2019

Предметни професор:
проф. др Владимир Миловановић

Садржај

Поставка задатка

Кроз поставку задатка дефинисано је да је потребно направити програм који за задату црно-белу слику креира ту исту слику само у боји. Програм као улаз прима црно-белу слику и помоћу генеративно супарничких мрежа креира слику у боји.

Опис и коришћење апликације

За израду програма користили смо interactive python notebook у Google Colab-у. За почетак мора да се скине zip фајл img_color_data у коме се налазе слике за тренирање и тестирање мреже. У том фајлу налази се 3000 различитих слика 150x150 неке су у боји, а неке су црно-беле. Кликом на дугме Runtime отвара се падајући мени где треба изабрати опцију Run all и тада ће се покренути програм. Само тренирање мреже траје око 3h што је доста тако да корисник мора да буде стрпљив док се цела мрежа истренира. Када се тренирање заврши слике које смо издвојили за тестирање бивају приказане и то у редоследу црно-бела слика, генерисана слика у боји, права слика како би смо лепо видели разлику између генерисане слике и праве слике.

Опис делова програма са самим изворним кодом

Прво морамо да се повежемо са Google Drive-ом.

```
from google.colab import drive
drive.mount('/content/drive')
```

Слика 1. Повезивање са Google Drive-ом

Затим unzip-ујемо фолдер img_color_data.zip како бисмо учитали слике.

```
!unzip /content/drive/MyDrive/img_color_data.zip
```

Слика 2. Учитавање слика

Додатни пакети коришћени при изради програма су:

- PIL
- numpy
- tensorflow
- os
- matplotlib
- sklearn

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

Слика 3. Коришћени пакети

Од учитаних слика користимо 2500 слика и сваку од њих resize-ујемо на величину 120x120 и пребацујемо их у црно-беле слике, а затим праве слике које су у боји и црно-беле слике нормализујемо тако што све вредности поделимо са 255. Затим све те слике делимо на тренинг и тест скуп и величина batch-a је 64 слика.

```
img_size = 120
master_dir = 'data'
number_of_images=2500

x=list()
y=list()

for image_file in os.listdir(master_dir)[:number_of_images]:
  rgb_image = Image.open(os.path.join(master_dir, image_file)).resize((img_size,img_size))

  # Normalizing the RGB images
  rgb_img_array = (np.asarray(rgb_image)) / 255
  # RGB to B/W
  gray_image = rgb_image.convert('L')
  # Normalizing the B/W images
  gray_img_array = (np.asarray(gray_image).reshape((img_size,img_size,1))) / 255

  # Appending both x and y lists
  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)

dataset = tf.data.Dataset.from_tensor_slices((train_x,train_y))

batch_size = 64

dataset = dataset.batch(batch_size)
```

Слика 4. Припремање података за тренинг и тестирање

Направљене су посебно функције за имплементацију генератора и дискриминатора

```
def gen():
  model = tf.keras.Sequential()

  model.add(tf.keras.Input(shape = (img_size,img_size,1)))

  model.add(tf.keras.layers.Conv2D(16, kernel_size=(5,5),strides=(1,1), padding='valid'))
  model.add(tf.keras.layers.LeakyReLU(alpha=0.3))
  model.add(tf.keras.layers.Conv2D(32, kernel_size=(3,3),strides=(1,1), padding='valid'))
  model.add(tf.keras.layers.LeakyReLU(alpha=0.3))
  model.add(tf.keras.layers.Conv2D(32, kernel_size=(3,3),strides=(1,1), padding='valid'))
  model.add(tf.keras.layers.LeakyReLU(alpha=0.3))


  model.add(tf.keras.layers.Conv2D(32, kernel_size=(5,5),strides=(1,1), padding='valid'))
  model.add(tf.keras.layers.LeakyReLU(alpha=0.3))
  model.add(tf.keras.layers.Conv2D(64, kernel_size=(3,3),strides=(1,1), padding='valid'))
  model.add(tf.keras.layers.LeakyReLU(alpha=0.3))
  model.add(tf.keras.layers.Conv2D(64, kernel_size=(3,3),strides=(1,1), padding='valid'))
  model.add(tf.keras.layers.LeakyReLU(alpha=0.3))

  model.add(tf.keras.layers.Conv2D(64, kernel_size=(5,5),strides=(1,1), padding='valid'))
  model.add(tf.keras.layers.LeakyReLU(alpha=0.3))
  model.add(tf.keras.layers.Conv2D(128, kernel_size=(3,3),strides=(1,1), padding='valid'))
  model.add(tf.keras.layers.LeakyReLU(alpha=0.3))
  model.add(tf.keras.layers.Conv2D(128, kernel_size=(3,3),strides=(1,1), padding='valid'))
  model.add(tf.keras.layers.LeakyReLU(alpha=0.3))

  model.add(tf.keras.layers.Conv2D(128, kernel_size=(3,3), strides=(1,1), activation='tanh', padding='same'))

  model.add(tf.keras.layers.Conv2DTranspose(128, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.Conv2DTranspose(128, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.Conv2DTranspose(64, kernel_size=(5,5),strides=(1,1), padding='valid', activation='relu'))

  model.add(tf.keras.layers.Conv2DTranspose(64, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.Conv2DTranspose(64, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.Conv2DTranspose(32, kernel_size=(5,5),strides=(1,1), padding='valid', activation='relu'))

  model.add(tf.keras.layers.Conv2DTranspose(32, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.Conv2DTranspose(32, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.Conv2DTranspose(3, kernel_size=(5,5),strides=(1,1), padding='valid', activation='relu'))

  return model
```

Слика 5. Архитектура генератрора

```
def disc():
  model = tf.keras.models.Sequential()

  model.add(tf.keras.layers.Conv2D(32, kernel_size=(7,7),strides=(1,1), padding='valid', activation='relu', input_shape=(img_size,img_size,3)))
  model.add(tf.keras.layers.Conv2D(32, kernel_size=(7,7),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D())

  model.add(tf.keras.layers.Conv2D(64, kernel_size=(5,5),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.Conv2D(64, kernel_size=(5,5),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D())

  model.add(tf.keras.layers.Conv2D(128, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.Conv2D(128, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D())

  model.add(tf.keras.layers.Conv2D(256, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.Conv2D(256, kernel_size=(3,3),strides=(1,1), padding='valid', activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D())

  model.add(tf.keras.layers.Flatten())
  model.add(tf.keras.layers.Dense(512, activation='relu'))
  model.add(tf.keras.layers.Dense(128, activation='relu'))
  model.add(tf.keras.layers.Dense(16, activation='relu'))
  model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

  return model
```

Слика 6. Аритектура дискриминатор

Направљене су функције које враћају вредности функције трошкова за генератор и дискриминатор. Функције трошкова су две бинарне cross entropy-је и средња квадратна грешка за дискриминатор и генератор респективно.

```
cross_entropy = tf.keras.losses.BinaryCrossentropy()

def disc_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output) , real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

mse = tf.keras.losses.MeanSquaredError()

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

gen_optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)
disc_optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)

generator = gen()
discriminator = disc()
```

Слика 7. Функције губитака за дискриминатор и генератор

Такође овде смо дефинисали и оптимизаторе за обе мреже. Наравно, користили смо Адам.

Искоритистили смо функцију за један тренинг корак, односно за један Batch, са документације Tensorflow-a. Модификовали смо је мало јер смо променили функције трошкова за наш проблем

```
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)


    g_loss = gen_loss( generated_images , real_y )
    d_loss = disc_loss(real_output,generated_output)



  print(g_loss)
  print(d_loss)

  gradients_gen = gen_tape.gradient(g_loss, generator.trainable_variables)
  gradients_disc = disc_tape.gradient(d_loss, discriminator.trainable_variables)

  gen_optimizer.apply_gradients(zip(gradients_gen, generator.trainable_variables))
  disc_optimizer.apply_gradients(zip(gradients_disc, discriminator.trainable_variables))
```

Слика 9. Тренинг корак за један batch

Дефинисано је број епоха на 150 итерација. За тих 150 итерација позивамо train_step методу.

```
num_epochs = 150

for i in range(1, num_epochs+1 ):
    print( i )
    for (x,y) in dataset:
        # (x,y) is batch
        print(x.shape)
        train_step(x,y)
```

Слика 10. Тренирање

Сада је потребно утврдити како наш програм ради на тестном скупу. На подацима које никада није видео. То ћемо увидети коришћењем пакета matplotlib.

```
y = generator(test_x[ : ]).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 Image', 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('Real Image', fontsize=16)
  plt.imshow( image )

  plt.show()
```

Слика 11. Приказ црно-белих, генерисаних и правих слика у боји

Литература

1. Tensorflow документација за генеративно-супарничке мреже https://www.tensorflow.org/tutorials/generative/dcgan
2. Курс на coursera-и, Конволуционе Неуронске Мреже https://www.coursera.org/learn/convolutional-neural-networks
3. Материјали са предмета Основа Дубоког Учења на Факултету Инжењерских Наука у Крагујевцу https://elektrotehnika.github.io/dl/